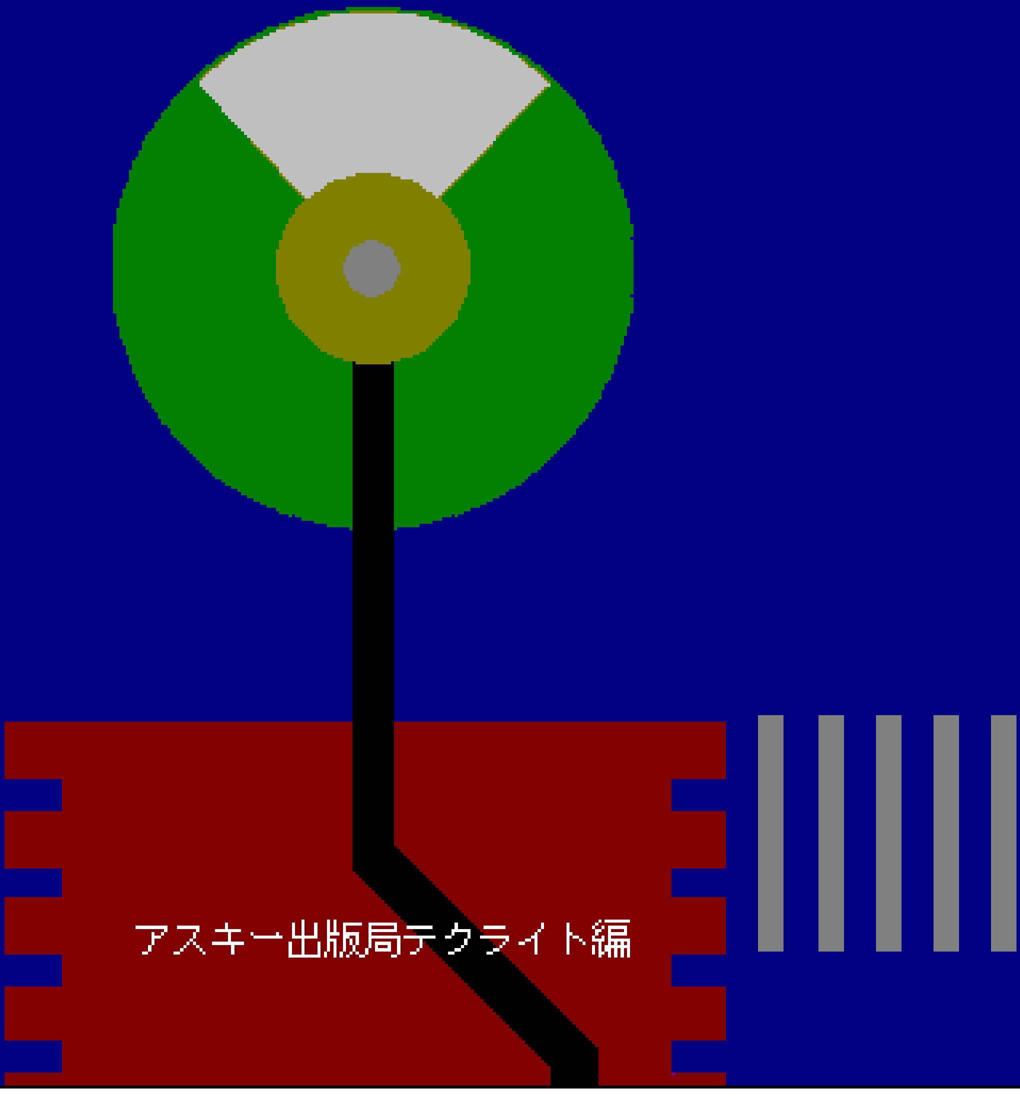
■ CD-ROM版 ■

PC-9800シリーズ テクニカルデータブック

MULTIMEDIA 編



第1部
マルチメディア概要

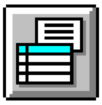
第1章

仕様概要

PC-9821, PC-98GSは従来のPC-9801にマルチメディアへ対応するためのハードウェアを付加した機種である. 新たに加わった機能には, グラフィックス関連と, サウンド関連がある. グラフィックス関連には, 多色発色, ビデオ入出力, 画面合成, スーパーインポーズなど, サウンド関連にはライン入出力, ミキシング, サンプリング, CD-DAなど, また従来からあったサウンド機能は強化されている.

これらの機能の関係を次の表に示す.

▼*機種ごとのマルチメディア機能一覧*



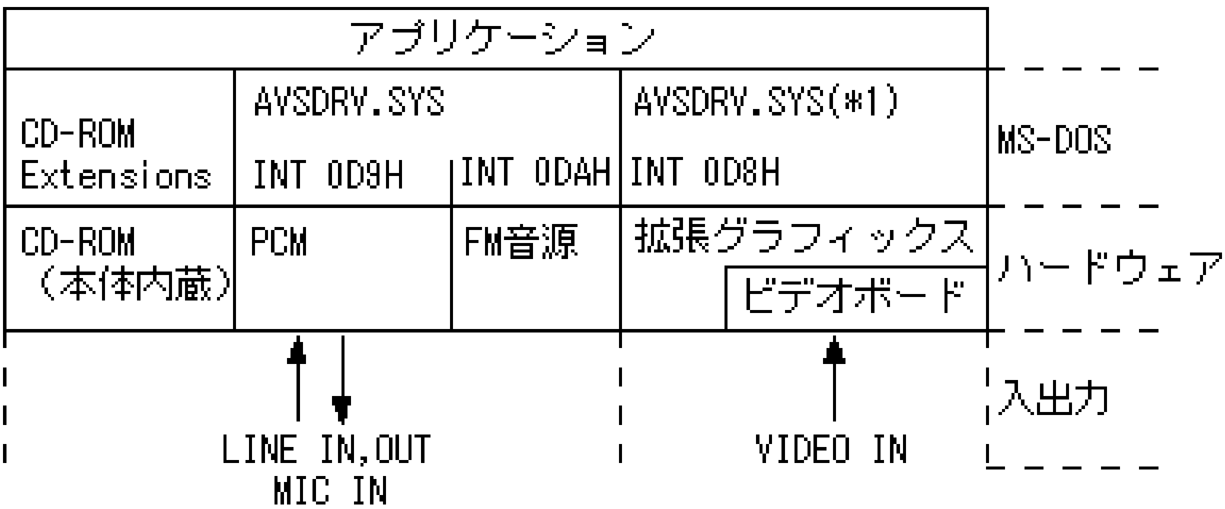
1.1　MS-DOSからの使用

新しく加わった機能を使用するために,グラフィックス関連とサウンド関連のそれぞれに“AVGDRV.SYS”,“AVSDRV.SYS”,“VIDEODRV.SYS”という名前のMS-DOS用のデバイスドライバが用意されている.これらの拡張ドライバは,EMSメモリに対応しており,メインメモリに入りきらないようなデータも扱うことができる.

これらのドライバは,MS-DOS Ver 5.0以降,およびEMS 4.0以降にのみ対応している.それ以外の環境では動作しないので注意が必要である.

また,ドライバの機能は機種により一部もしくはすべてがサポートされていない場合がある.各機種のカタログ,マニュアル等で仕様を確認後利用すること.

ドライバをアプリケーションから呼び出すにはソフトウェア割り込みを使用すること.



(*1)PC-98GSを除くPC-9801シリーズの機種では，ビデオボード（PC-9801-72）用ドライバとして”VIDEODRV.SYS”を利用する．

1.2　Microsoft Windowsからの使用

Microsoft Windows(以下Windows)は,Ver 3.1以降MME(Multi Media Extention)によってマルチメディアに対応しているため,特別なドライバを使うことなく拡張された機能を利用できる.そのためWindows用に開発されたマルチメディア対応のソフトウェアはそのまま使用できる.

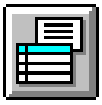
第2章

グラフィックス機能

2.1　表示機能

PC-9821，PC-98GSには，従来のPC-9800シリーズにはなかったビデオ機能が拡張されている．PC-9801においても拡張ビデオボードを追加することで同様な機能を得ることができる．

▼*機種によるグラフィックス機能の違い*



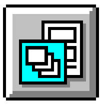
■グラフィックス画面

●PC-9821シリーズ，PC-9801

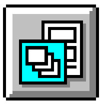
PC-9821Aシリーズ，Xシリーズ，98MULTiでは，98グラフィックス(16色)に加えて256色表示機能が追加されている．またPC-9821Aシリーズ，98MULTiでは，画面モードとして，今までの640×400の解像度の他に640×480が追加されている．

PC-9801-72使用時には，ビデオ画面を表示するためのVRAMをフレームバッファとして使用できる(ただし，メモリアクセスに時間がかかる)．このときには解像度が640×200/400，同時1677万色の画面となる．

▼*グラフィックス画面の表示優先順位(PC-9821)*



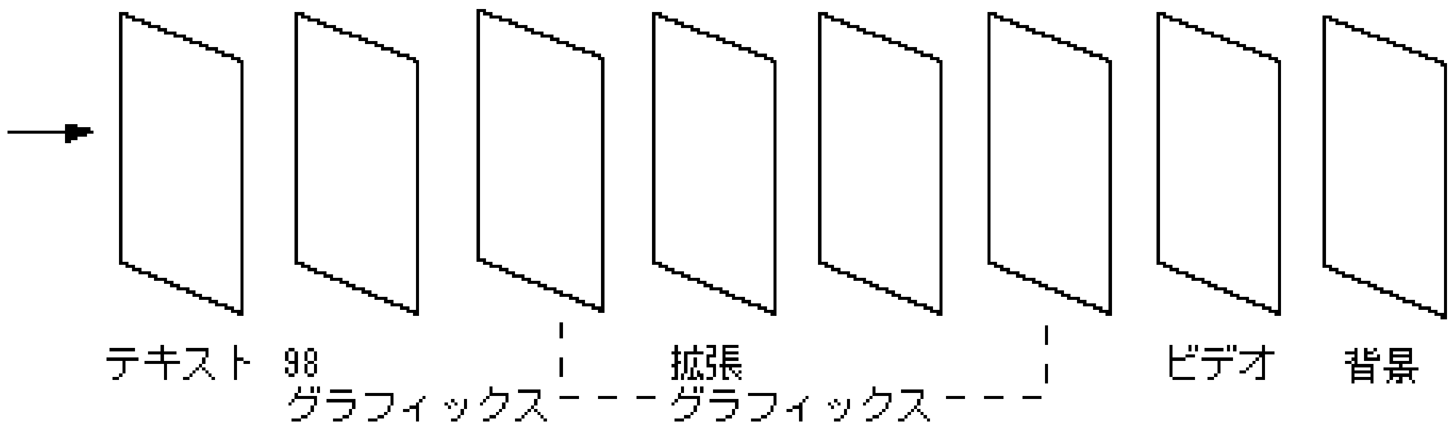
▼*グラフィックス画面の表示優先順位(PC-9801)*



●PC-98GS

PC-98GSでは98グラフィックスに加えて，4面の拡張グラフィックス画面，ビデオ画面，背景面が追加されている．このうちテキスト，98グラフィックス，拡張グラフィックスについては重ね合わせるときの順序を自由に設定できる．ビデオ画面と背景面については変更できず，常に他の画面の裏になる．

拡張グラフィックス画面については0から4枚の間で何枚使用するかを変更できる．この拡張グラフィックスでは，256色，65,000色，1677万色を同時表示することができる．



・テキスト，98グラフィックス，拡張グラフィックスの各画面(最大6画面)を任意に並べ替え設定可能．
・拡張グラフィックス画面は設定により0～4枚に可変(色数と解像度との兼ね合いにより変化する)．
・ビデオ画面はオプションのビデオボード(PC-98GS-02)使用時に利用可能．

拡張グラフィックス画面で使用できる解像度には，以下のようなものがある．

画面モード一覧
(フレームバッファ水平サイズ×垂直サイズ×ピクセルサイズ)

▼*拡張グラフィックス画面の画面モード一覧*

2.2　ハードウェア

PC-9821(Bシリーズを除く)，PC-98GSでは拡張グラフィックスのため専用のVRAMが用意されている．このVRAMは通常は表示されないようになっているが，後述する拡張グラフィックスドライバを呼び出すことで，表示，描画等の操作を行うことができる．

このVRAMのメモリと画面上の表示の対応は，同時発色する色数によって変化し，下の表のようになっている．

発色数	1ピクセルあたりのデータ長
256色	1バイト(8ビット)
65,000色	1ワード(16ビット)
1677万色	RGBそれぞれに1バイト(8ビット)ずつ，RGBは別々のプレーン

CPUのメモリ空間にはVRAMすべてを入れるだけの余裕がないため，その一部だけをCPUのメモリ空間にマッピングできるようになっている．このためにCPUのメモリ空間に32KBずつの2つの拡張グラフィックスウィンドウと呼ばれるマッピングするための領域が確保されている．

それぞれのウィンドウには拡張VRAMの別々の領域をマッピングできる．

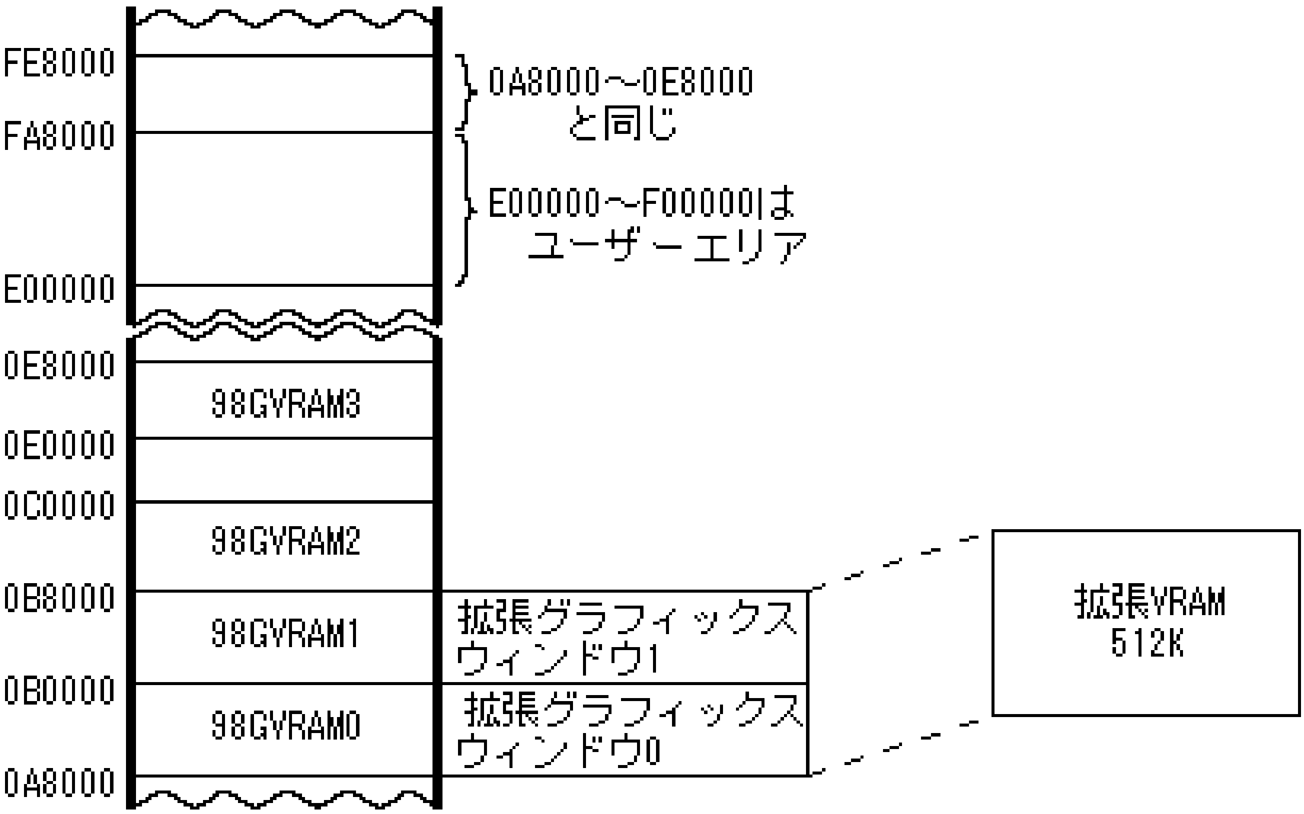
●拡張グラフィックスVRAMとグラフィックスドライバの関係

▼*拡張G-VRAMとグラフィックスドライバの関係(メモリマップ)*

▼*拡張G-VRAMとグラフィックスドライバの関係(CRTとG-VRAM)*

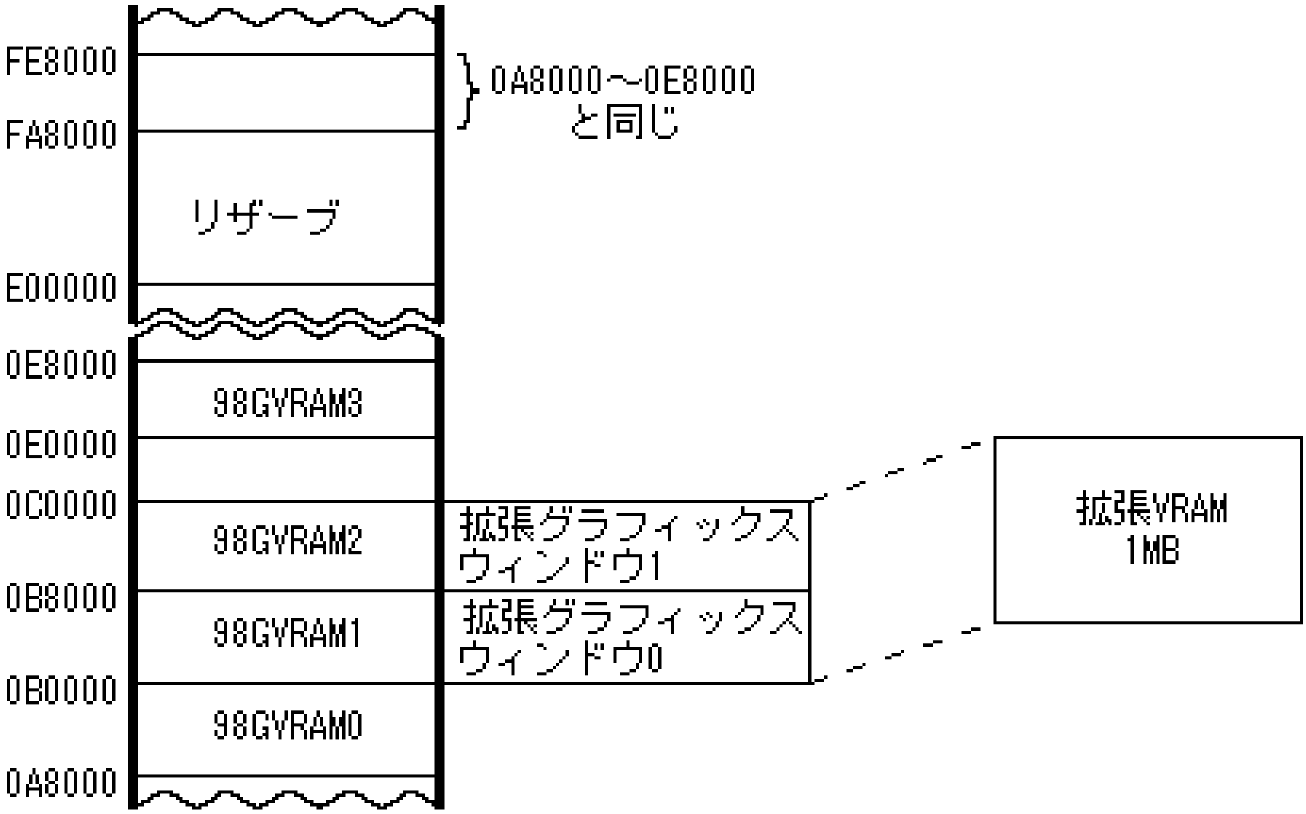
●PC-9821

拡張グラフィックスのために512KBのVRAMが用意されている．拡張グラフィックスドライバを呼び出すことによりVRAMがCPUのメモリ空間にマッピングされ，CPUから直接操作することができるようになる．



●PC-98GS

1MBのVRAMが拡張グラフィックスのために用意されている．このVRAMはCPUのメモリ空間にはマッピングされておらず，拡張グラフィックスドライバをとおしてしか使用できない．



■ビデオ画面

オプションのビデオボードを取付けることで，外部からのビデオ入力をデジタイズして取り込み，それを本体の画面と重ね合わせて表示することができる．ビデオ信号はRGB各8ビット，つまり1677万色で取り込まれる．

このボードには以下のような機能がある．

ビデオ信号の変換

ビデオ信号をアナログRGB信号へ同期するように変換する．

ビデオ画面の縮小

ビデオ画面の大きさを縮小し，指定された位置で表示する．

ビデオ画面のフリーズ

連続して動作しているビデオ信号の取り込みを停止する．

画面合成

本体の画面とビデオ画面を合成する．このときに画面の順序は設定できる．

透明色指定

指定された色を透明色として扱うことで，その色で描画された部分は下の画面が透けてみえるようにする．

モノクロ指定

指定された色をモノクロで表示する．

ユーザコントロール

画質，コントラスト，ブライト，色合い，色の濃さをソフトウェアで制御する．

フィールドメモリの読み書き

ビデオ信号が取り込まれるフィールドメモリの内容をI/Oポートを通して読み書きする．

本体画面とのドット位置補正

本体の機種による表示位置の違いを補正する．

本体画面サンプリング位相調整

本体のD/Aコンバータのばらつきに対応する．

●PC-9821/PC-9801+ビデオボード

　ビデオボード(PC-9801-72)を取付けることにより本体の画面にビデオ入力をスーパーインポーズすることができる．ビデオ画面は常に一番下の画面となる．

●PC-98GS

　専用のビデオボード(PC-98GS-02)を取付けることにより，外部からのビデオ入力と本体の画面を合成して表示することができる．
　ビデオ入力は65,000色または1677万色のいずれかのモードでデジタイズされてから合成される．

■背景面

　PC-98GSにのみある画面である．
　すべての画面の一番後に表示される．1677万色から選択した1色で画面全体を塗りつぶすように表示する．すなわち他の画面は，その色の前に重なって表示される．

■特殊表示機能

　PC-98GSのみにある機能である．

●透明色指定

　透明な色を表示することにより，その部分は重なり合った下の画面が透けて表示される．この機能により，上の画面で枠を作りその中を透明色でくりぬき，任意の形の下の画面への窓を作ることができる．

●矩形マスク

　四角いマスク画面を作って，画面と画面の間に挟み込むことにより，マスク画面の内側だけに下の画面が表示される．透明色指定を使っても同じことができるが，透明色指定では一画面を使ってしまう．矩形マスクを使うことで画面を無駄にすることなく下の画面を表示することができる．マスクの大きさと位置は，水平には1ピクセル単位，垂直には2ピクセル単位で指定できる．

●演算合成表示

　重ね合わされた画面のうち，一番手前にあるものとその次のものとの間で演算を行いその結果を表示する．この演算の結果は表示だけに影響し，VRAMには影響を与えない．

透明合成
　一番目の画面の黒の部分にだけ二番目の画面を表示する．

半透明
　両方の画面の平均値を表示する．

加算合成
　両方の画面を加算した結果を表示する．

フィルタ処理
　二番目の画面を一番目の画面の状態に合わせて表示する．表示されるのは“(二番目の画面)-NOT(一番目の画面)”となる．

論理演算
　各種論理演算を二つの画面の間で行いその結果を表示する．

フェードイン/フェードアウト
　両方の画面のそれぞれに数の加減算を行い，その結果を加算合成した結果を表示する．

■ビデオエフェクト

PC-98GSのみにある機能である．

ビデオ入力をデジタイズした結果をリアルタイムに処理し，その結果を表示することでビデオエフェクトを行うものである．このビデオエフェクトはSGP(Super Graphic Processor)がすべての処理を行い，CPUは使用されない．

拡張グラフィックス画面のうち2画面を使用する．片方の画面へビデオ入力をデジタイズし，その画面に対してSGPが処理を行いながら他方の画面へと転送することでこの処理を行っている．処理結果の転送された画面だけを表示することでビデオエフェクトのかかった画面のみが表示される．

SGPが行える処理は次のとおりである．

表示画面クリア	ランダム圧縮処理	輪郭強調処理
部分画面クリア	残像化処理	高輝度部分減輝処理
カラー反転処理	格子抜き転送処理	ノイズ追加処理
部分カラー反転処理	ズーム転送	白黒化処理
左右鏡処理(左画面がベース)	ソフトフォーカス	白黒拡大縮小処理
左右鏡処理(右画面がベース)	千鳥状リバース	マスク付き白黒化処理1
上下鏡処理(上画面がベース)	輝度1.5倍転送	マスク付き白黒化処理2
上下鏡処理(下画面がベース)	輝度0.5倍転送	256色処理
左右反転処理	揺らぎ転送	256色拡大縮小処理
上下反転処理	鏡反射処理	256階調白黒化処理
部分左右反転処理	奇数行左右反転処理	台形転送(下側圧縮)
部分上下反転処理	0度回転処理	台形転送(下側圧縮，　上下反転)
分割画面処理	90度回転処理	台形転送(上側圧縮)
部分縮小処理	180度回転処理	台形転送(上側圧縮，　上下反転)
モザイク処理	270度回転処理	台形転送(右側圧縮)
部分モザイク処理	マスク転送処理	台形転送(右側圧縮，　左右反転)
球面処理	部分論理演算処理	台形転送(左側圧縮)
ランダム転送処理	輪郭抽出処理	台形転送(左側圧縮，　左右反転)

2.3　ソフトウェア

■概要

拡張されたグラフィックス機能を使うためには，拡張グラフィックスドライバ(AVGDRV.SYS)を組み込む必要がある．

ただし，このドライバはMS-DOS Ver 5.0以降にのみ対応しており，それ以前のバージョンでは動作しない．EMSを使用する場合には，EMS 4.0以上に対応したEMSドライバが必要であり，その場合には0C000Hセグメントから始まる2ページ以上の連続した物理ページが必要である．

このドライバによって拡張ビデオの機能をアプリケーションプログラムから使用することができる．

デバイスドライバは，次の行をCONFIG.SYSに加えるか，ADDDRVコマンドの定義ファイルに書いてコマンドを実行することによってシステムに組み込まれる．

DEVICE=AVGDRV.SYS [/E] [/C]

/EはEMSを使用する場合にのみ必要であり，使わない場合には省略してかまわない．

/CはNDP(浮動小数点プロセッサ)を使用する場合に必要となる．NDPがない場合，または使わない場合には省略する．

デバイスが組み込まれるさいには以下のようなメッセージが表示される．

・正常組み込み時

「AVグラフィックスドライバが使用できます」
「EMSを使用します」(EMSを使用する場合のみ)
「数値データプロセッサを使用します」(NDPを使用する場合のみ)

・組み込み中止時(パラメータにエラーがあった場合)
「パラメータの指定に誤りがあります」
「AVグラフィックスドライバは使用できません」

・組み込み中止時(動作に必要なハードウェアがなかった場合)
「AV機能が無効です」
「AVグラフィックスドライバは使用できません」

正常に組み込まれた場合には，“AVGDRV$$”というデバイス名がシステムに組み込まれる．

■ドライバの呼び出し

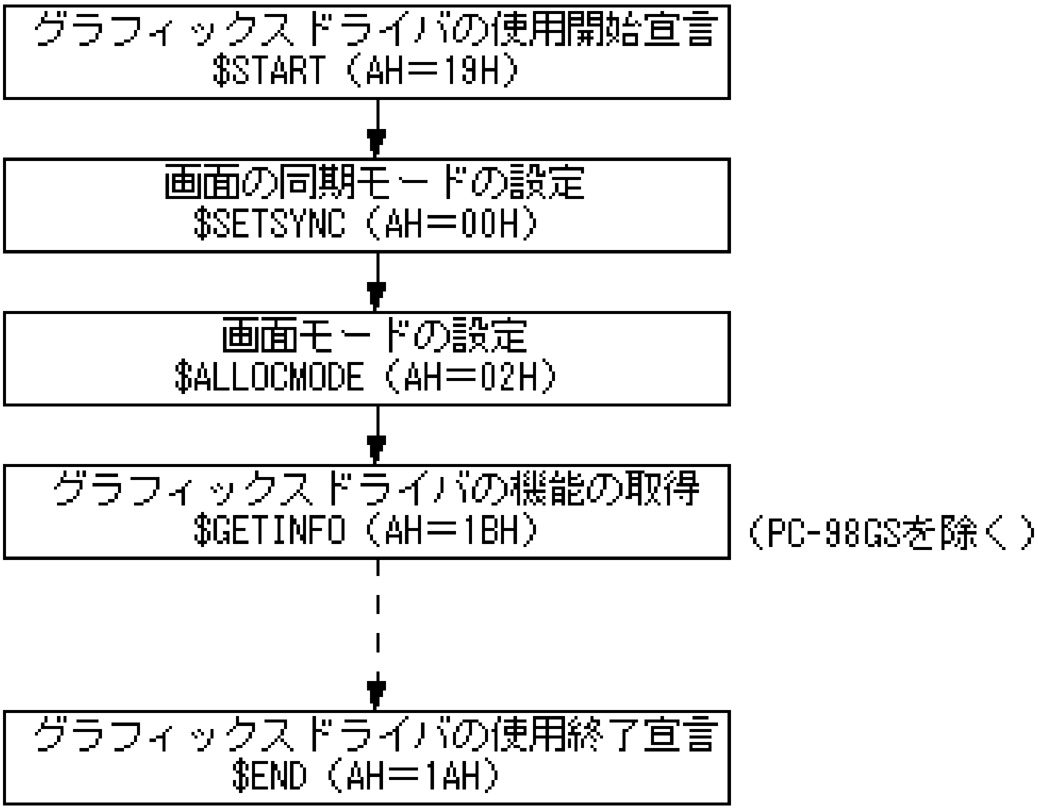
ドライバを呼び出すには，ソフトウェア割り込みを使用する．使用する割り込みを以下に示す．

割り込み	INT 0D8H

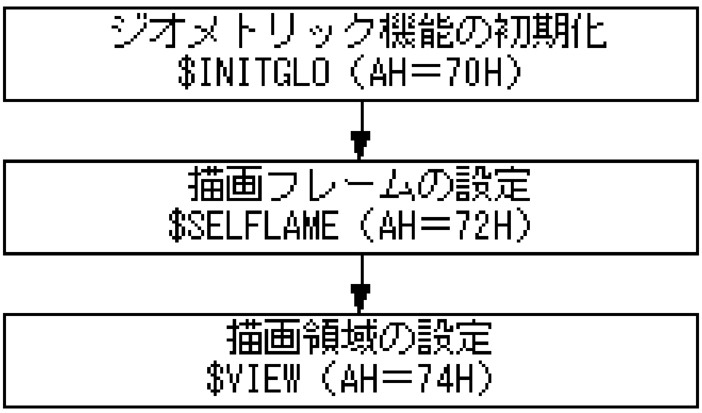
ドライバは呼び出したアプリケーションのスタックをそのまま使用するため，呼び出す側で少なくともスタックに256バイトの余裕をもたせる必要がある．
ソフトウェアからドライバが組み込まれているかどうかは，デバイス名“AVGDRV$$”がオープンできるかどうかで判断できる．

ドライバを使用する場合には以下のことに注意する．
アプリケーションプログラムはドライバを使用するときには，最初に「グラフィックスドライバの使用開始宣言(AH = 19H)」を呼び出し，最後に「グラフィックスドライバの使用終了宣言(AH = 1AH)」を呼び出さなくてはならない．

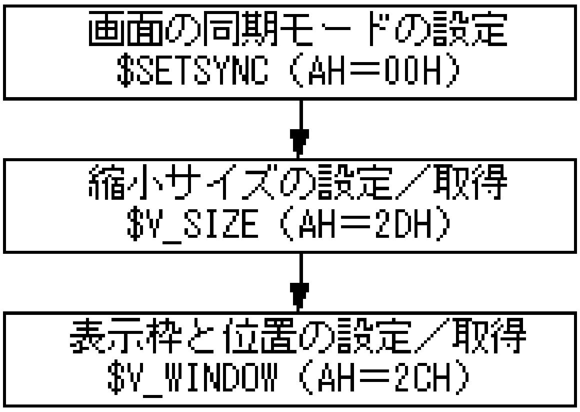
以下にドライバを使用するときの手順を示す.



これに加えて，ジオメトリック系のファンクションを使用する場合は以下の手順でファンクションを呼び出すこと.



水平周波数の設定を変更した上でビデオ系ファンクションを使用する場合には，次のような手順でドライバを呼び出し再設定を行わねばならない．



■エラーコード

すべてのドライバのファンクションは，エラーが発生したときにはキャリーフラグを1にし，エラーコードをAHレジスタにセットしてアプリケーションプログラムへ戻る．

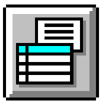
エラーコード	意味
80H 未満	各ファンクション参照
80H	致命的なエラー（ハードウェアトラブルなど）
81H	二重呼び出し（回復可能）
0FFH	未サポートファンクション呼び出し

第3章

サウンド機能

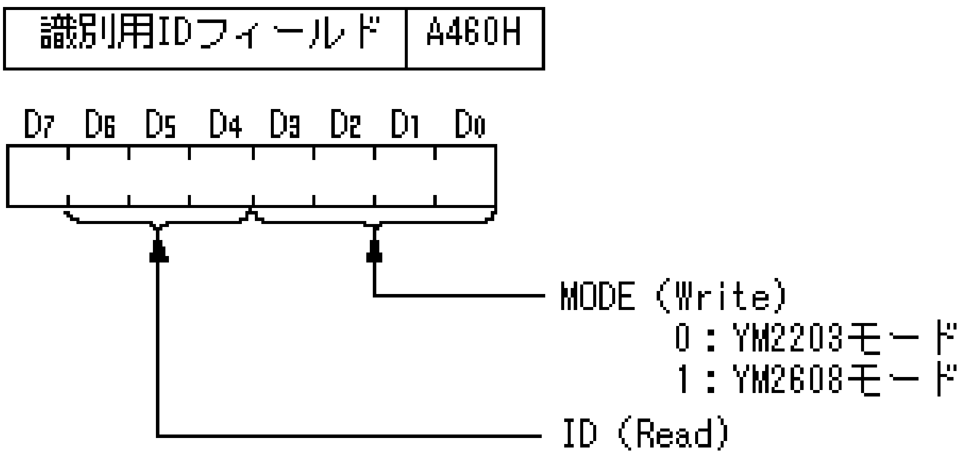
　PC−9821，PC−98GSおよびオプションのサウンドボード(PC−9801−73，PC−9801−86)には，今までのサウンド機能に加えて新しい機能が追加されている．

▼*機種によるサウンド機能の違い*



3.1　ハードウェア

サウンド機能を識別するためのI/Oポートが用意されている.



■拡張FM/SSG音源

これまで使われていた音源LSIのYM2203のかわりに上位互換のYM2608が使われており，これまでのFM3和音+SSG3和音の互換演奏モードに加え，FM6和音+SSG3和音+リズム6和音の新しい拡張演奏モードが加わっている.

音源LSI	FM 音源	SSG 音源	リズム音源
2203 互換	3 和音（モノラル）	3 和音（モノラル）	なし
2608 拡張	6 和音（ステレオ）	3 和音（モノラル）	6 和音（ステレオ）

拡張モードのFM音源とリズム音源では，ある音をRチャネルとLチャネルのそれぞれについて，出力するかどうか独立に設定することができ，これによってステレオ出力を行っている．互換モードやSSG音源では音がRチャネルとLチャネルに同時に出力される．

リズム音源には以下の音が含まれている．

Rim Shot	Cymbal
Tom	Snare Drum
High Hat	Bass Drum

■I/Oアドレス

I/Oアドレスは，表のようになっている．

音源LSI	アドレス・ステータス	データ
2203 互換	0188H	018AH
2608 拡張	018CH	018EH

■PCM/DSP機能

▼*機種によるサンプリング音源の違い*

今までのサウンド機能に追加になったのは以下のような機能である.

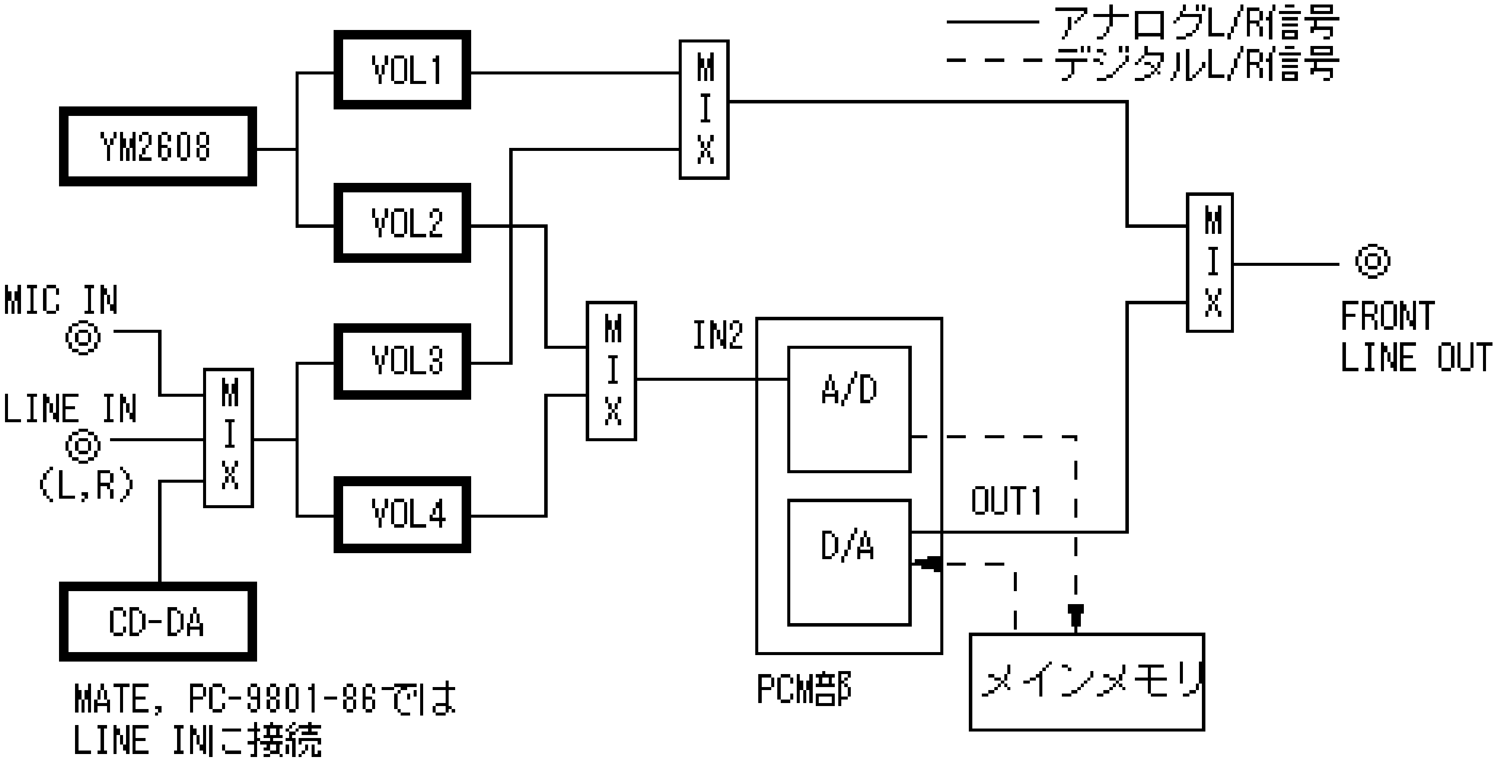
・内蔵FM/SSG/リズム音源とPCM出力のライン入力，CDの演奏のミキシング
・PCMによるミキシングした音のデータの取り込みや再生
・DSPを使ってのミキシングした音の加工

PCM/DSPへの入力としては，以下の三種類が使用できる.

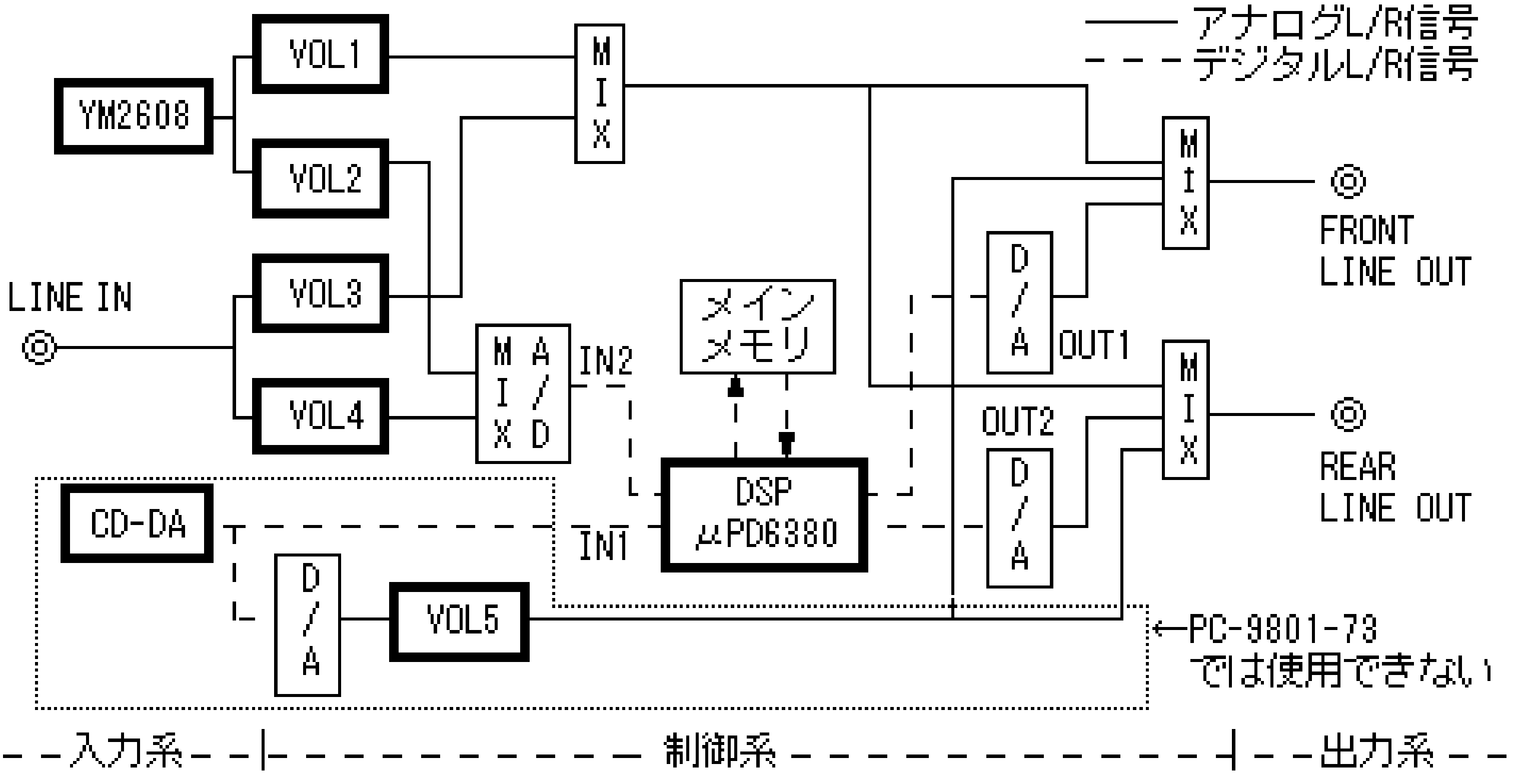
・FM/SSG/リズム音源
・LINE入力
・マイク入力(PC-9821，PC-9801-86)
・CD-DA(PC-98GS，PC-9821)

これらのうちモノラルであるSSG音源およびマイク入力については，内部でステレオ化されてから，他の音源と同様に扱われる.
これらからの入力は，それぞれ独立に電子ボリュームによって音量を調節され，アナログ出力とPCM/DSPへの入力に振り分けられる.
PCM/DSPへの入力は，A/D(アナログ/デジタル)コンバータによっていったんデジタル(PCM)データに変換されてから，PCM/DSPで処理される．その出力はD/A(デジタル/アナログ)コンバータによって再びアナログへ戻されてから元のアナログ出力とミキシングされて出力となる.
ただし，PC-98GSではCD-DAはCDのデジタル出力をそのままPCM/DSPの入力へ接続して処理するようになっている(他の機種ではライン入力からのアナログ入力となる).

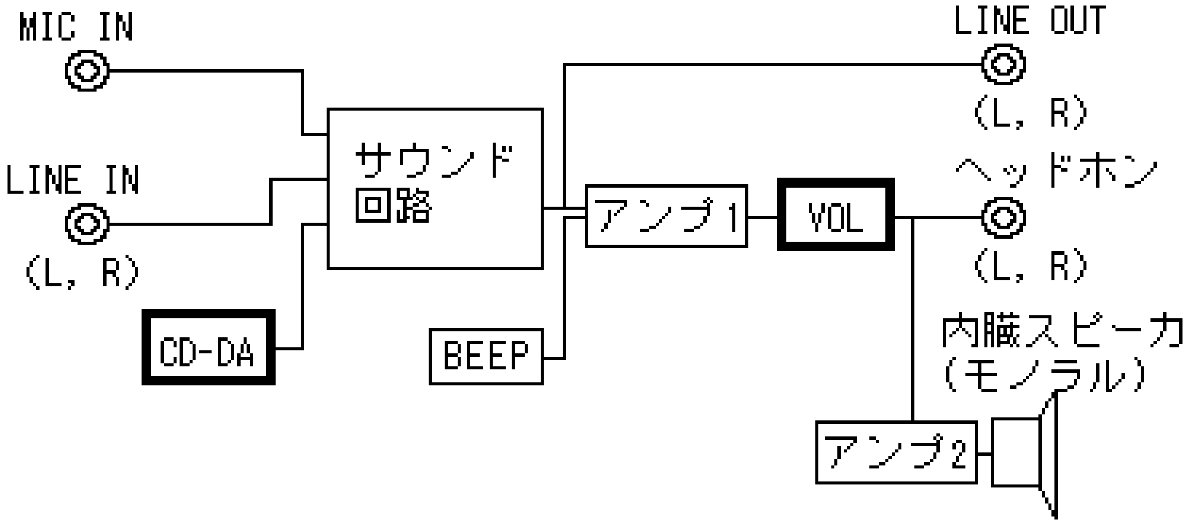
●PC-9821/PC-9801-86



●PC-98GS/PC-9801-73



●PC-9821Xシリーズ



■電子ボリューム

電子ボリュームでは，電気的にアナログ音量を調節する．
入力された音量の0%から100%まで10%単位で変化させることができる．電子ボリュームは4個(PC-98GSでは5個)用意されており，以下のものがある．

・FM/SSG/リズム音源とLINE出力の間(VOL1)
・FM/SSG/リズム音源とA/Dコンバータの間(VOL2)
・LINE入力とLINE出力の間(VOL3)
・LINE入力とA/Dコンバータの間(VOL4)
・CD-DAとLINE出力の間(VOL5)(PC-98GSのみ)

VOL1，VOL2，VOL3は，入力されたアナログデータをそのままLINE出力へ出力する音量の調節を行う． これに対してVOL3とVOL4はアナログデータのA/Dコンバータへの入力の音量を調節する．
以上のように，入力された音量の調節は可能であるが，PCMからの出力の音量調節を行うことはできない．PCMからの出力の音量を変えたい場合には，PCMへ入力する時点で音量調整を行っておく必要がある．

■A/Dコンバータ，D/Aコンバータ

アナログとデジタル，デジタルとアナログの間でデータの変換を行う.

PCMはCPUが直接これらとの間でデータのやり取りを行うことで実現している. 取り込んだデータはメインメモリまたはEMSメモリに置くことができ，ディスクへ書き出して保存することも可能である. 逆にディスクから読み込んでそれを再生することもできる.

サンプリング周波数

44.10KHz*，33.08KHz*，22.05KHz,16.54KHz,
11.03KHz，8.27KHz，5.52KHz，4.13KHz
*PC-98GSでは不可

量子化ビット数

8ビット，16ビット

入力チャネル

モノラル(R，L)，ステレオ

ただし上のサンプリング周波数は，機種のCPUや動作状態，他の周辺機器からの割り込みなどによって実用となる周波数が変化する.

状況によってはPCMによる取り込み・再生時にデータの抜け，化けが発生しうる. そのような場合には，サンプリング周波数を下げる必要がある. このため機種によってはそのCPU性能に応じてドライバでサンプリング周波数に対して制限を加えているものがある.

■DSP(Digital Signal Processor)

　デジタルデータの処理を専門に行う高速なプロセッサである.
　デジタル化された音のデータを処理することによって，さまざまな楽器でいうエフェクターと同じような効果を与えることができる.
　この効果は，入力された音に対してリアルタイムに発生する．そのため，もとのアナログの音とDSPで処理した音をともに出力することで，複雑な処理を行うこともできる．DSPで行える機能は以下の通りである．これらのうちからいずれか一つだけを実行できる.

エコー	ステレオコーラス
ローパスフィルタ	ダブリング
ハイパスフィルタ	フランジャ
グラフィックスイコライザ	ビブラート
リバーブ	アンサンブル
ホール効果	ピッチシフタ
音像定位	サラウンド
定位補正	ADPCM
パンニング	

　これらのうちADPCMは，DSPで8ビットもしくは16ビットでサンプリングしたデータを処理し4ビットの差分に変換してCPUへ渡したり，逆を行ったりするものである.
　DSPの性能はCPUや割り込みの状態によらず変化しないので，PCMにあるような制限はDSPによる効果においては存在しない.

3.2　ソフトウェア

■概要

拡張されたサウンド機能を使うためには，拡張サウンドドライバ(AVSDRV.SYS)を組み込む必要がある．

ただし，このドライバはMS-DOS Ver5.0以降にのみ対応しており，それ以前のバージョンのMS-DOSでは動作しない．EMSを使用する場合には，EMS 4.0以上に対応したEMSドライバが必要であり，その場合には0C000Hセグメントから始まる4ページ以上の連続した物理ページが必要である．

このドライバによって拡張FM/SSG音源とPCM/DSP機能をアプリケーションプログラムから使用することができる．

デバイスドライバは，次の行をCONFIG.SYSに加えるか，ADDDRVコマンドの定義ファイルに書いてコマンドを実行することによってシステムに組み込まれる．

DEVICE=AVSDRV.SYS[/E]

/EはEMSを使用する場合にのみ必要であり，使わない場合には省略してかまわない．デバイスが組み込まれるさいには以下のようなメッセージが表示される．

・正常組み込み時

「AVサウンドドライバが，使用できます」

「EMSを使用します」(EMSを使用する場合のみ)

・組み込み中止時(パラメータにエラーがあった場合)

「パラメータの指定に誤りがあります」

「AVサウンドドライバは使用できません」

・組み込み中止時(動作に必要なハードウェアがなかった場合)
　　　　「AV機能が無効です」
　　　　「AVサウンドドライバは使用できません」

正常に組み込まれた場合には，“AVSDRV$$”というデバイス名がシステムに組み込まれる．

■ドライバの呼び出し

ドライバを呼び出すには，ソフトウェア割り込みを使用する．使用する割り込みを以下に示す．

割り込み	PCM/DSP	INT 0D9H
	FM音源	INT 0DAH

ドライバは呼び出したアプリケーションのスタックをそのまま使用するため，呼び出す側で少なくともスタックに256バイトの余裕をもたせる必要がある．

ソフトウェアからドライバが組み込まれているかどうかは，デバイス名“AVSDRV$$”がオープンできるかどうかで判断できる．

■PCM/DSP機能の初期化

FM/SSGドライバとPCMドライバは同一の領域を利用している．そのため，他のPCM/DSPファンクションを利用する前には本ファンクションを実行し，FM/SSGドライバのファンクション「ドライバの初期化(INT 0DAH, AH=00H)」もしくは「拡張モードでのドライバの初期化(INT 0DAH, AH=00H)」のいずれも実行して，両方を初期化しておく必要がある．

第4章

Microsoft Windows

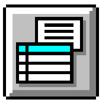
4.1　マウスドライバ

Windows 3.1において，マウスを利用するMS-DOSアプリケーションをウィンドウ内実行を行う場合には，以下の点に注意する必要がある．

■マウスドライバの種類

MS-DOS Ver3.3xもしくはVer5.0x，Windows 3.1において添付されているマウスドライバには以下のものがある．

▼*各OSに添付のマウスドライバと機能一覧*



■マウスドライバの利用方法

Windows 3.1においては，MS-DOSアプリケーションとWindowsアプリケーションのデータ連携を容易にするため，MS-DOSアプリケーションのウィンドウ内表示/実行機能がサポートされている．しかしこの場合，Windows 3.1自身のマウスと競合が発生するため，MOUSE.COM対応(MOUSE Ver7.02のAPIを使用)のアプリケーションしかマウスを使用することができない．

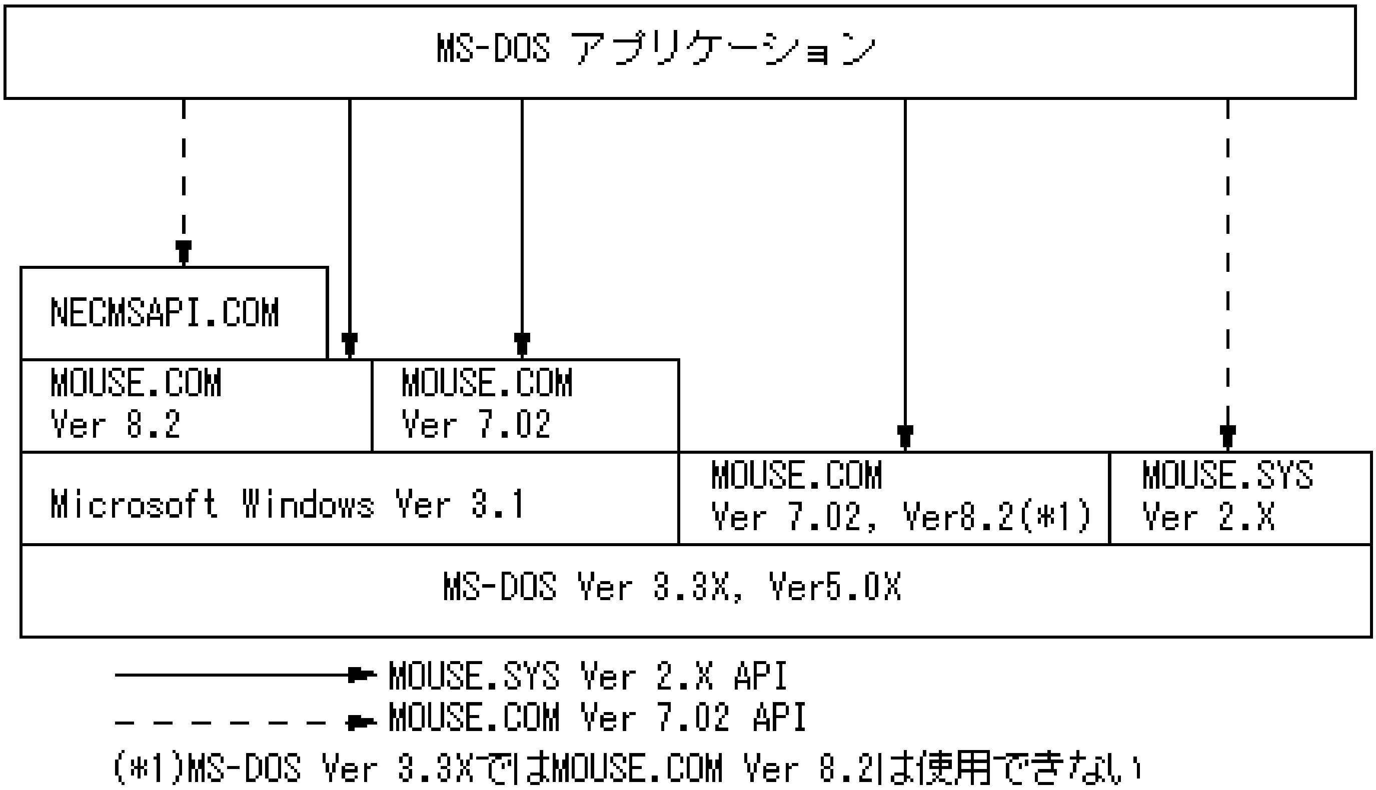
そのため，MOUSE.COM Ver8.2には，MOUSE Ver2.X APIのエミュレーションを

行うことにより，MOUSE.SYS対応のアプリケーションのウィンドウ内実行における マウスの使用を可能とする機能がある．この機能を利用するためには，Windows 3.1に添付のNECMSAPI.COMというコマンドを利用する必要がある．

●利用手順

①Windows 3.1において，MS-DOSウィンドウを起動する．
②MS-DOSのコマンドラインからWindows 3.1添付のMOUSE.COMを実行する．
③MS-DOSのコマンドラインからNECMSAPI.COMを実行する．
　これによりMOUSE Ver2.X APIがエミュレートされる．
④アプリケーションを実行する．

アプリケーション終了後，NECMSAPI.COMを再度実行すると，NECMSAPI.COMは常駐解除される．



MOUSE.COM Ver8.2の使用方法については，Windows 3.1に添付のREADME.DOCを参照すること．

■MOUSE.COM対応の利点

ＭＳ－ＤＯＳアプリケーションはＭＯＵＳＥ．ＳＹＳに対応することにより，MS-DOS Ver3.X，Ver5.0Xの両方に対応することができる．

しかし，MS-DOS Ver5.0Xの利用を前提とした場合は，MOUSE.COM対応とすると以下の利点がある．

・NECMSAPI.COMを実行しなくてもWindows 3.1でのMS-DOSウィンドウ内でマウスを使用できる．
・テキストカーソルを使用できる．
・マウスポインタが画面の背景の色の影響を受けないため常に同じ色で表示される．

4.2 MICROSOFT WINDOWS SOUND SYSTEM

■概要

Windows上で動作するMICROSOFT WINDOWS SOUND SYSTEM Ver1.0にはDDEサーバの機能を持つテキストリーダ(TTS.EXE)が添付されている.

DDEインターフェイスを利用することにより，テキストデータをリアルタイムに読み上げるWindowsアプリケーションを作成することが可能となっている.

■DDEメッセージ

DDEクライアントアプリケーションは，以下の5つのDDEメッセージをテキストリーダ(DDEサーバ)に送信することができる.

WM_DDE_POKE
　テキストデータをテキストリーダに送信する．テキストリーダは送られたテキストデータを読み上げる.

WM_DDE_ADVISE
　テキストリーダが，読み上げ終了時にDDEクライアントアプリケーションに終了の通知を行うようにする.

WM_DDE_EXECUTE
　“STOP”(停止)，“PAUSE”(一時停止)，“RESUME”(再開)のコマンドをテキストリーダに送信する.

WM_DDE_INITIATE
　DDE通信の初期化を行う.

WM_DDE_TERMINATE
　DDE通信を終了する.

■DDEプロトコル

テキストリーダでは，DDEクライアントとの通信にあたり，以下の3つのパラメータを定義している.

・Application:TtsServer
・Topics:Tts(& System)
・Item:Read

■Visual Basicでの使用例

Visual Basicでは，DDEによるアプリケーション間のリンクを簡単に行うことができるため，テキストの読み上げを行うアプリケーションが容易に開発できる.

ここでは，Visual Basicを使用したテキストリーダとのDDE通信のためのサンプルプログラムを示す.

●テキストの読み上げ

テキストデータを読み上げるためには，LinkPokeメソッドを使用する.

以下の例では，“SPEECH MENU”をクリックすると，DDEクライアントは“Speech Text”テキストボックスにあるテキストデータをテキストリーダに送信する.

```
Sub SPEECH_Click 0
 ConstNONE=0, MANUAL=2, NOTIFY=3

 SpeechText.LinkMode=NONE
 SpeechText.LinkTopic="ttsserver|tts"
 SpeechText.LinkItem="read"
 SpeechText.LinkMode="MANUAL"
 SpeechText.LinkPoke
 SpeechText.LinkMode="NONE"
End Sub
```

●読み上げ終了の確認

読み上げ終了の確認をDDEクライアント側で行うには，DDE通信を開始する際にLinkModeプロパティでNOTIFYをセットする．

以下の例では，“SPEECH MENU”をクリックすると，DDEクライアントは“Speech Text”テキストボックスにあるテキストデータをテキストリーダに送信する．そして，テキストリーダは読み上げ終了後，DDEクライアントに対して終了通知を返す．

```
Sub SPEECH_Click 0
 ConstNONE=0, MANUAL=2, NOTIFY=3

 SpeechText.LinkMode=NONE
 SpeechText.LinkTopic="ttsserver|tts"
 SpeechText.LinkItem="read"
 SpeechText.LinkMode="NOTIFY"
 SpeechText.LinkPoke
End Sub

Sub SpeechText_LinkNotify()
 MsgBox "read finished"
 SpeechText.LinkMode=NONE
End Sub
```

●読み上げの停止

テキストの読み上げを中止するには，LinkExecuteメソッドを使用して，コマンド文字列“stop”を送信する．

以下の例では，“STOP MENU”をクリックすると，DDEクライアントは“stop”をテキストリーダに送信する．テキストリーダは“stop”を受信すると読み上げを停止する．

```
Sub STOP_Click 0
 ConstNONE=0, MANUAL=2, NOTIFY=3

 SpeechText.LinkMode=NONE
 SpeechText.LinkTopic="ttsserver|tts"
 SpeechText.LinkItem="read"
 SpeechText.LinkMode="MANUAL"
 SpeechText.LinkExecute "stop"
 SpeechText.LinkMode="NONE"
End Sub
```

●読み上げの一時停止

テキストの読み上げを一時停止するには，LinkExecuteメソッドを使用して，コマンド文字列“pause”を送信する．

以下の例では，“PAUSE MENU”をクリックすると，DDEクライアントは“pause”をテキストリーダに送信する．テキストリーダは“stop”を受信すると読み上げを一時停止する．

```
Sub PAUSE_Click ()
 ConstNONE=0, MANUAL=2, NOTIFY=3

 SpeechText.LinkMode=NONE
 SpeechText.LinkTopic="ttsserver|tts"
 SpeechText.LinkItem="read"
 SpeechText.LinkMode="MANUAL"
 SpeechText.LinkExecute "pause"
 SpeechText.LinkMode="NONE"
End Sub
```

●読み上げの再開

テキストの読み上げを再開するには，LinkExecuteメソッドを使用して，コマンド文字列“resume”を送信する．

以下の例では，“RESUME MENU”をクリックすると，DDEクライアントは“resume”をテキストリーダに送信する．テキストリーダは“resume”を受信すると読み上げを再開する．

```
Sub RESUME_Click 0
 ConstNONE=0,MANUAL=2,NOTIFY=3

 SpeechText.LinkMode=NONE
 SpeechText.LinkTopic="ttsserver|tts"
 SpeechText.LinkItem="read"
 SpeechText.LinkMode="MANUAL"
 SpeechText.LinkExecute "resume"
 SpeechText.LinkMode="NONE"
End Sub
```

第2部
拡張グラフィックス
ドライバ

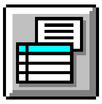
第1章

拡張グラフィックスドライバサポートファンクション概要

以下のファンクションをサポートする.

●画面設定系ファンクション一覧(INT 0D8H)

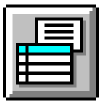
▼*画面設定系ファンクション*



●ビデオ系ファンクション一覧(INT 0D8H)

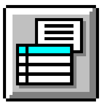
ビデオボード(PC-9801-72，PC-98GS-02)が必要.

▼*ビデオ系ファンクション*



●ジオメトリック系ファンクション一覧(INT 0D8H)

▼*ジオメトリック系ファンクション*



第2章

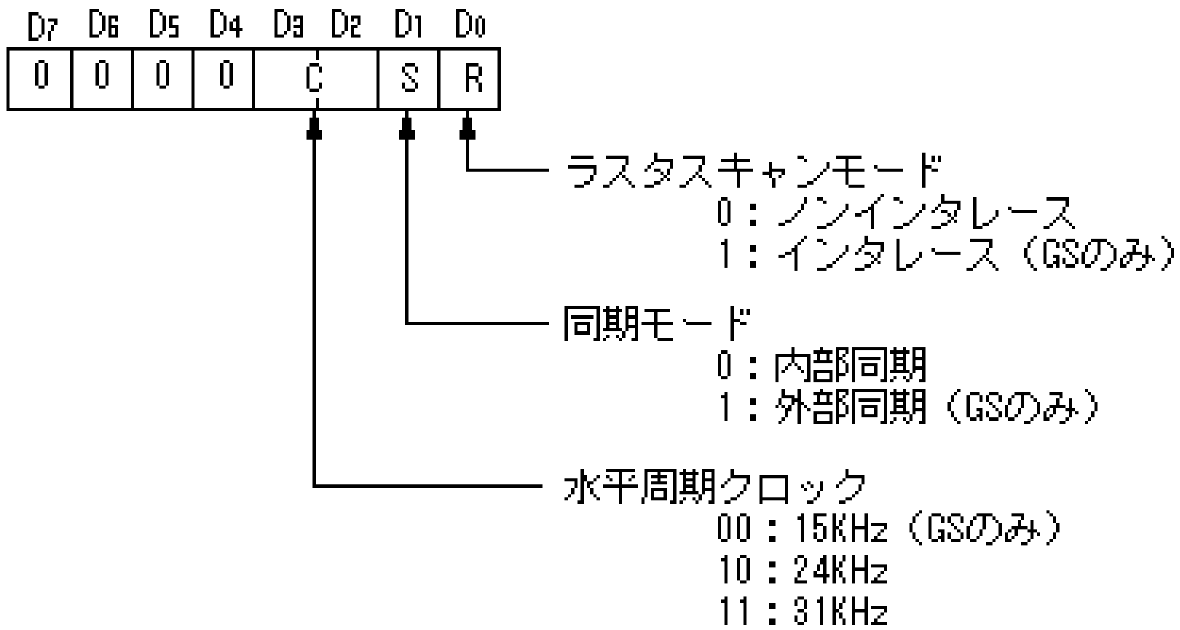
画面設定系ファンクション

$SETSYNC	AH=00H	
画面の同期モードの設定		

入　力

AH ＝ 00H

AL ＝ 同期モード



BH = 同期タイプ

D7	D6	D5	D4	D3	D2	D1	D0
0	0	L		0	0	R	

テキスト行数（R）
00：20行
01：25行
10：30行

98グラフィックスライン数（L）
00：200ライン（上半分）
01：200ライン（下半分）
10：400ライン
11：480ライン

BL = テキスト表示

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	T

テキスト表示（T）
0：テキスト表示禁止
1：テキスト表示許可

出　力

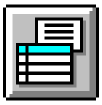
Carry = 同期ステータス
0 ： 正常設定
1 ： エラー

AH = エラーステータス
1 ： パラメータ異常
2 ： 同期タイミングオーバーラン

機　能

画面の同期モードの設定を行う.
同期タイプは同期周波数にしたがって以下の制限がある.

▼*同期タイプの同期周波数による制限*



注　意

本ファンクション実行後は，テキスト，98グラフィックスGDCが初期化されるので，各GDCの設定をしている場合は再設定が必要である．テキスト画面は，画面表示停止となる．98グラフィックス画面は，設定前の状態どおり，表示停止/許可となる.

$ASKSYNC	AH=01H	

画面の同期モードの取得

入　力

AH = 01H

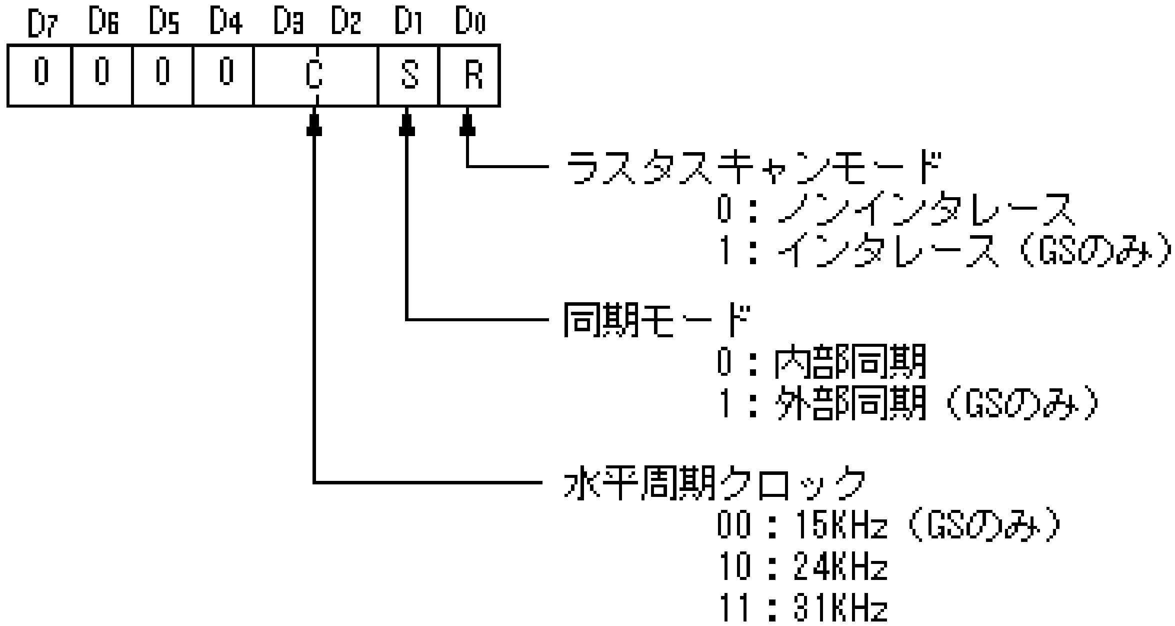
出　力

Carry = 同期ステータス

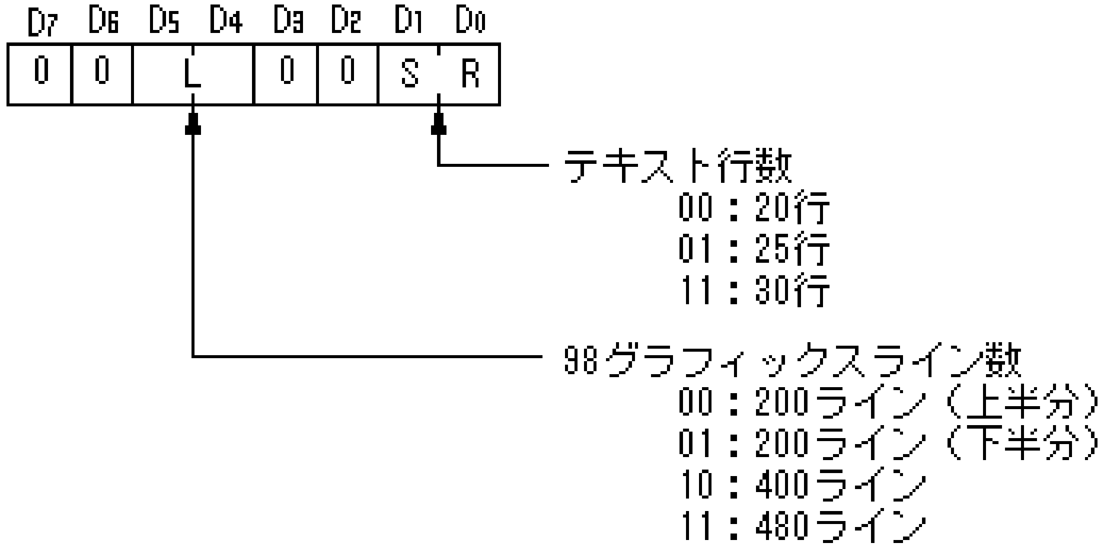
0 ： 正常設定

1 ： 「画面の同期モードの設定(AH=00H)」による同期モードの設定が行われていない.

AL = 同期モード



BH = 同期タイプ



機　能

画面の同期モードを取得する

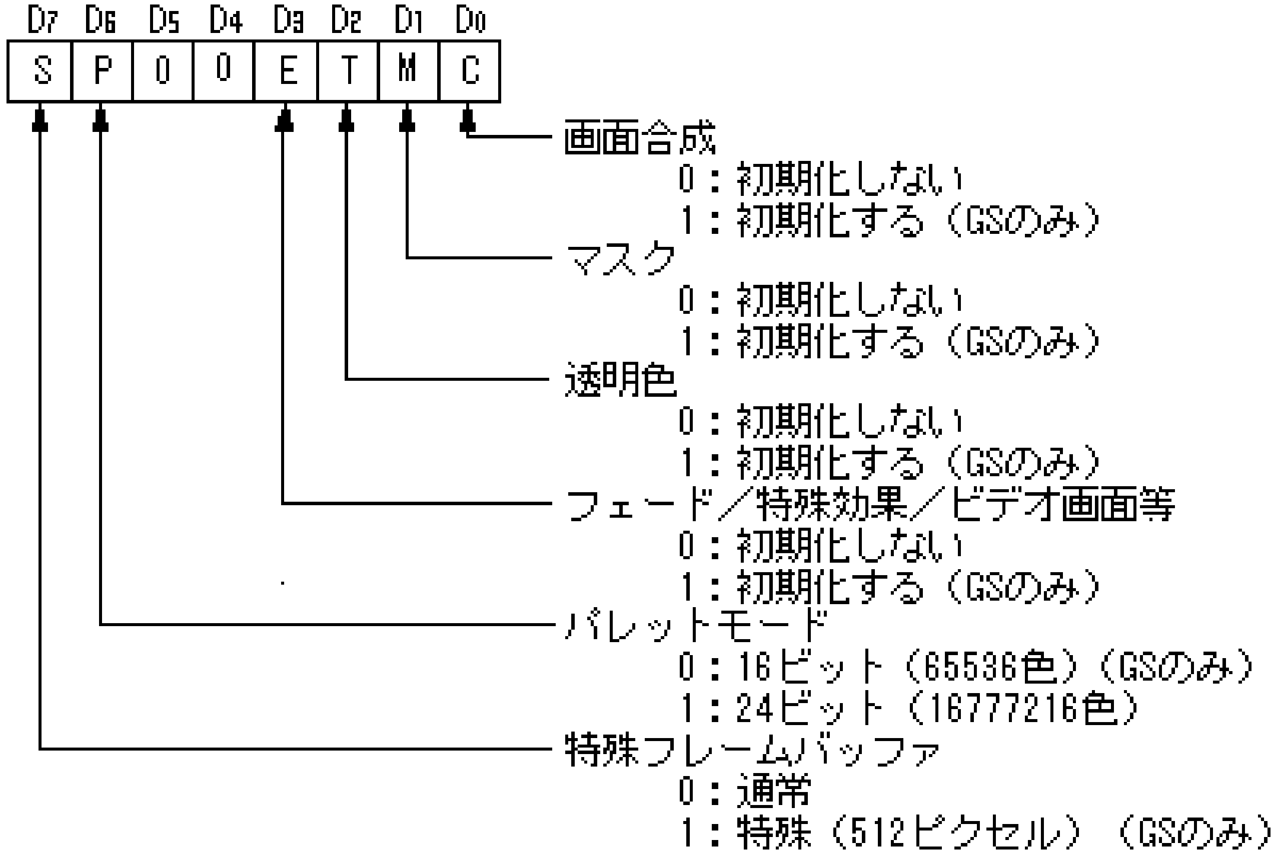
$ALLOCMODE	AH=02H	

画面モードの設定

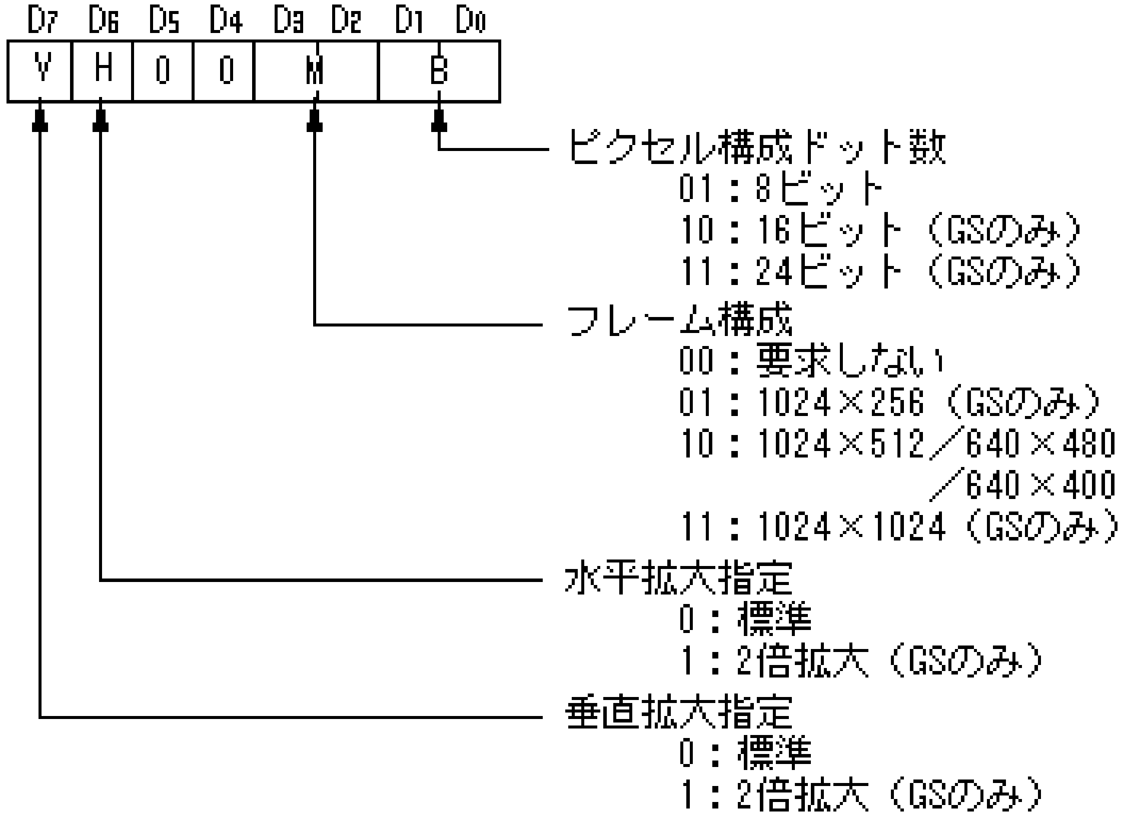
入　力

AH = 02H

AL = 画面モード



BH ＝ 要求画面モード1



BL ＝ 要求画面モード2（構成は要求画面モード1と同様）
（GS/400ライン時のみ）

CH ＝ 要求画面モード3（構成は要求画面モード1と同様）
（GSのみ）

CL ＝ 要求画面モード4（構成は要求画面モード1と同様）
（GSのみ）

出　力

Carry = フレーム獲得状況

0 ： 全画面獲得

1 ： 獲得不能画面あり

BH = フレーム0獲得画面モード

D7	D6	D5	D4	D3	D2	D1	D0
V	H	0	0	M		B	

B（D1, D0）：ピクセル構成ドット数
- 00：8ビット（16ビットパレット）（GSのみ）
- 01：8ビット（24ビットパレット）（GSのみ）
- 10：16ビット（GSのみ）
- 11：24ビット（GSのみ）

M（D3, D2）：フレーム構成
- 00：獲得不能
- 01：1024×256（GSのみ）
- 10：1024×512／640×480／640×400
- 11：1024×1024（GSのみ）

H（D6）：水平拡大指定
- 0：標準
- 1：2倍拡大（GSのみ）

V（D7）：垂直拡大指定
- 0：標準
- 1：2倍拡大（GSのみ）

BL = フレーム1獲得画面モード(構成はフレーム0獲得画面モードと同様)
(GS/400ライン時のみ)

CH = フレーム2獲得画面モード(構成はフレーム0獲得画面モードと同様)
(GSのみ)

CL = フレーム3獲得画面モード(構成はフレーム0獲得画面モードと同様)
(GSのみ)

機　能

BH，BL，CH，CLにより要求された画面モードから表示モードを確定する．

アロケート(獲得)可能な画面について，水平/垂直拡大，カラーパレットのモードを設定する．

アロケート不能な画面に対してはフレーム構成として00を返し，水平/垂直拡大，カラーパレットのモード設定は行わない．

特殊フレームバッファモードが設定されている場合は，24ビット/ピクセル画面では横幅が1/2になり縦が2倍になる(GSのみ)．

このとき，表示幅が512ドットの範囲になるようにマスクが設定される．

ただし，水平拡大が指定されている場合にはこの限りではない(GSのみ)．

注　意

「画面フレームの初期化(AH=04H)」，「パレットの初期化(AH=09H)」(256色のみ)は，本ファンクションでは設定されない．

8ビット/ピクセル画面は，パレットモードにより獲得できる画面の数が異なる．

$ASKMODE	AH=03H	

画面モードの取得

入　力

AH = 03H

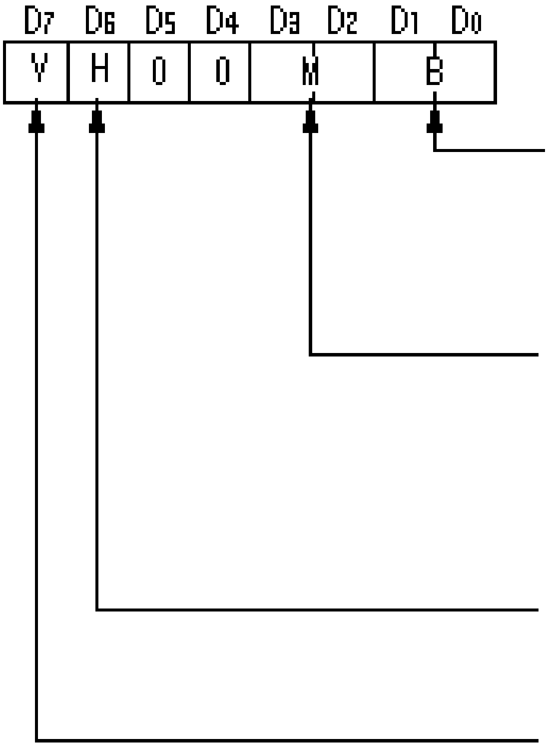
出　力

Carry = 実行ステータス

0 ： 正常

1 ： 「画面モードの設定(AH=02H)」による画面モードの設定が行われていない

BH = フレーム0画面モード



ピクセル構成ドット数
- 00：8ビット（16ビットパレット）（GSのみ）
- 01：8ビット（24ビットパレット）
- 10：16ビット（GSのみ）
- 11：24ビット（GSのみ）

フレーム構成
- 00：獲得不能
- 01：1024×256（GSのみ）
- 10：1024×512／640×480／640×400
- 11：1024×1024（GSのみ）

水平拡大指定
- 0：標準
- 1：2倍拡大（GSのみ）

垂直拡大指定
- 0：標準
- 1：2倍拡大（GSのみ）

BL = フレーム1画面モード(構成はフレーム0画面モードと同様)
(GS/400ライン時のみ)
CH = フレーム2画面モード(構成はフレーム0画面モードと同様)
(GSのみ)
CL = フレーム3画面モード(構成はフレーム0画面モードと同様)
(GSのみ)

機　能

現在の画面モードを取得する.

$INITFRAME	AH=04H	
画面フレームの初期化		

入　力

AH = 04H

出　力

Carry = 初期化ステータス
　　0 ： 初期化完了
　　1 ： エラー
AH = エラーステータス
　　1～3 ： 画面モードが不適切
　　　「画面モードの設定(AH=02H)」による画面モードの設定が行われていない

機　能

現在の画面モードに従って，以下に示すフレームバッファの初期値を設定する．

画面0のXオフセット	0	
画面0のYオフセット	0	
画面0の表示開始ライン	256/400/480	
画面0の表示ライン数	0	
画面1のXオフセット	0	GSもしくは9821 400ラインモードのみ
画面1のYオフセット	0	
画面1の表示開始ライン	0	
画面1の表示ライン数	0	
画面2のXオフセット	0 (-112(*1))	GSのみ
画面2のYオフセット	0	
画面2の表示開始ライン	0	
画面2の表示ライン数	0	
画面3のXオフセット	0	
画面3のYオフセット	0	
画面3の表示開始ライン	0	
画面3の表示ライン数	0	
その他のXオフセット	114	

(*1)特殊フレームバッファの場合

$SETFRAME	AH=05H	
画面フレームの設定		

入　力

AH = 05H
AL = 設定モード
　00H ： 画面0の設定
　01H ： 画面1の設定
　02H ： 画面2の設定(GSのみ)
　03H ： 画面3の設定(GSのみ)
　FFH ： 全画面の一括設定
BX = 画面フレームのXオフセット
CX = 画面フレームのYオフセット
DH = 画面フレームの表示開始ライン
DL = 画面フレームの表示ライン数
（BX～DL）…AL=00H～03Hのとき

ES:BX = 画面フレームフォームテーブル…AL=FFHのとき

テーブル構造

ES：BX→	画面0のXオフセット	それぞれ2バイトの値で設定する
	画面0のYオフセット	
	画面0の表示開始ライン	GS以外は0
	画面0の表示ライン数	
	画面1のXオフセット	GSもしくは9821 400ラインモードのみ
	画面1のYオフセット	
	画面1の表示開始ライン	
	画面1の表示ライン数	
	画面2のXオフセット	GSのみ
	画面2のYオフセット	
	画面2の表示開始ライン	
	画面2の表示ライン数	
	画面3のXオフセット	
	画面3のYオフセット	
	画面3の表示開始ライン	
	画面3の表示ライン数	

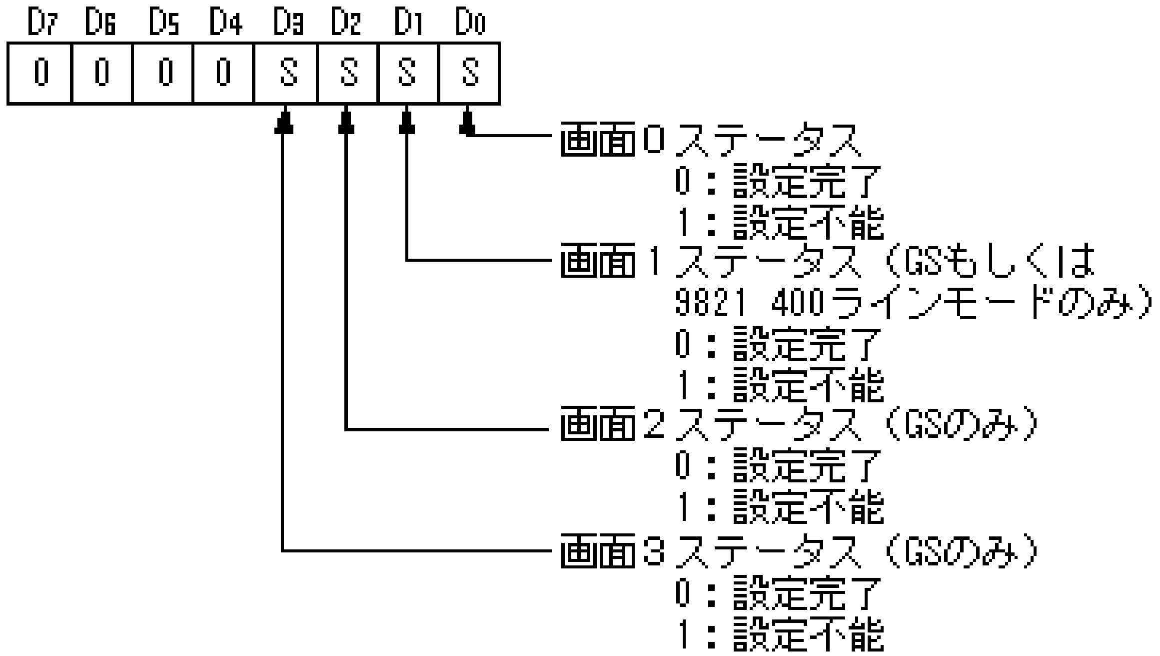
出　力

Carry = 設定ステータス

0 ： 正常設定

1 ： 「画面モードの設定(AH=02H)」による画面モードが設定されていない．画面番号エラー

AL = 設定ステータス　Carry = 0のとき



機　能

ALがFFH以外の場合，BX，CX，DH，DLにより指定された画面を定義する．

ALがFFHの場合，ES:BXで示されたフレームフォームテーブルの内容に従って，4つの画面を定義する．

表示ライン数は実際に表示されるライン数の半分の値を設定する．

表示ライン数として実際に表示可能なライン数以上を設定した場合は，表示可能な最大ライン数が設定される．

注　意

PC-9821でVRAMの先頭を表示画面左上にあわせる場合には以下のようにする.

画面フレーム(または画面1)のXオフセット =-48
画面フレーム(または画面1)のYオフセット =-40(400ライン時)
= 0(480ライン時)
画面フレーム(または画面1)の表示ライン数=200(400ライン時)
=240(480ライン時)

$ASKFRAME	AH=06H	
画面フレームの取得		

入　力

AH = 06H

AL = 獲得モード

00H ： 画面0の獲得

01H ： 画面1の獲得(GSもしくは9821 400ラインモードのみ)

02H ： 画面2の獲得(GSのみ)

03H ： 画面3の獲得(GSのみ)

FFH ： 全画面の一括獲得

ES:BX = 　画面フレームフォームテーブル…AL=FFHのとき

出　力

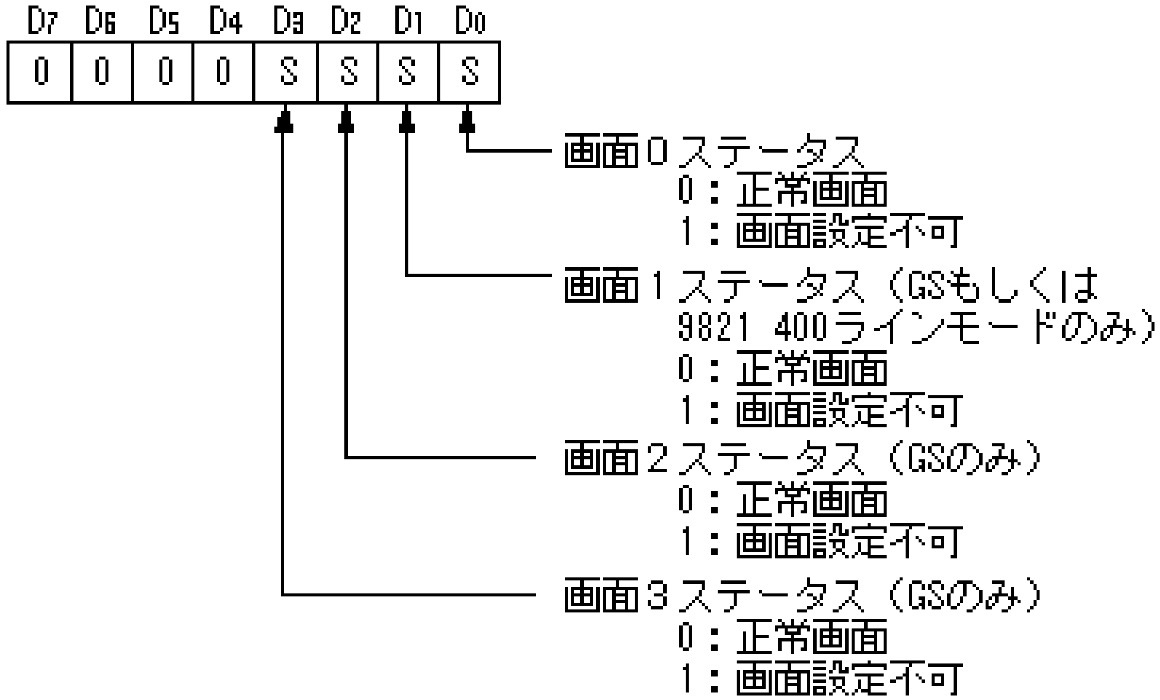
Carry = 実行ステータス

0 ： 正常設定

1 ： 「画面モードの設定(AH=02H)」による画面モードが設定されていない．画面番号エラー

AL ＝ 画面ステータス

（AL=FFH以外で呼ばれた場合は，該当しない画面は0を返す．）



BX ＝ 画面フレームのXオフセット
CX ＝ 画面フレームのYオフセット
DH ＝ 画面フレームの表示開始ライン
DL ＝ 画面フレームの表示ライン数

…AL ＝ 00H～03Hのとき

ES:BX = フレームフォームテーブル…AL = FFHのとき

テーブル構造

ES:BX→

項目	備考
画面0のXオフセット	それぞれ2バイトの値をとる
画面0のYオフセット	
画面0の表示開始ライン	GS以外は0
画面0の表示ライン数	
画面1のXオフセット	GSもしくは9821 200ラインモードのみ
画面1のYオフセット	
画面1の表示開始ライン	
画面1の表示ライン数	
画面2のXオフセット	GSのみ
画面2のYオフセット	
画面2の表示開始ライン	
画面2の表示ライン数	
画面3のXオフセット	
画面3のYオフセット	
画面3の表示開始ライン	
画面3の表示ライン数	

機　能

現在の画面フレームを取得する.
表示ライン数は実際に表示されるライン数の半分の値を返す.

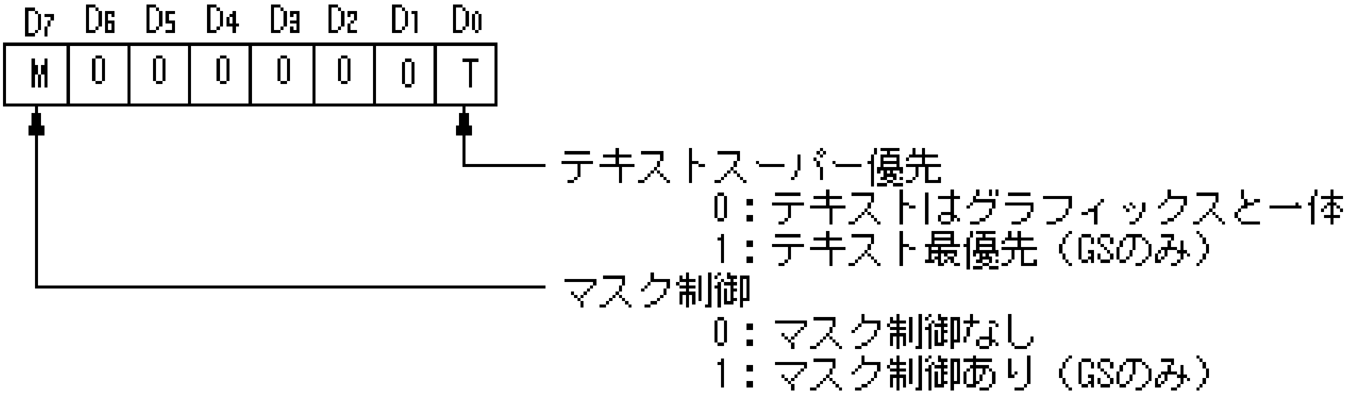
$COMPOSE	AH=07H	

画面優先順位の設定

入　力

AH = 07H

AL = テキスト/マスクモード



BH = 第1画面/第2画面指定

D7	D6	D5	D4	D3	D2	D1	D0
0	F			0	S		

第2画面指定
- 000：指定なし
- 001：カレント画面(*1)
- 010：画面0（GSのみ）
- 011：画面1（GSのみ）
- 100：画面2（GSのみ）
- 101：画面3（GSのみ）
- 110：ビデオ画面（ビデオボード使用時のみ）
- 111：設定禁止

第2画面指定
- 000：設定禁止（GSのみ）
- 001：カレント画面(*1)
- 010：画面0
- 011：画面1
- 100：画面2（GSのみ）
- 101：画面3（GSのみ）
- 110：ビデオ画面（ビデオボード使用時のみ）
- 111：設定禁止

(*1)9821の256色モードの場合以外は９８画面

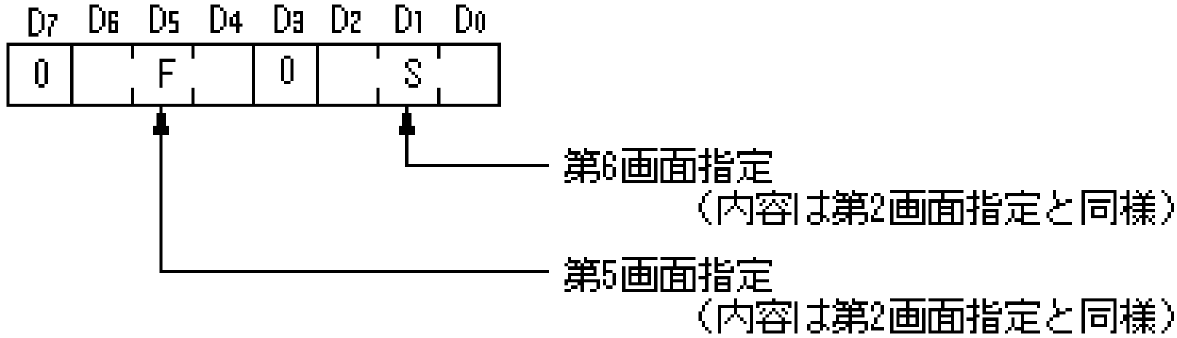
BL = 第3画面/第4画面指定(GS以外では必ず0にする)

D7	D6	D5	D4	D3	D2	D1	D0
0	T			0	F		

第4画面指定
（内容は第2画面指定と同様）

第3画面指定
（内容は第2画面指定と同様）

CH = 第5画面/第6画面指定(GS以外では必ず0にする)



出　力

Carry = 設定ステータス

0 ： 正常設定

1 ： 設定異常

(画面が重複指定されている．もしくは，「画面モードの設定(AH =02H)」による画面モード設定が行われていない)

機　能

優先順に指定された画面に従って，表示合成を行う．

テキストスーパー優先が1の場合は，テキストと98グラフィックス画面が分離し，テキスト画面が最優先画面になる．ただし，GS以外の機種では，常に「テキストとグラフィックス画面を一致」にしなければならない．

マスク制御なしの場合，優先順位を変更しても対応する画面マスクの順位は変わらない．したがって，本ファンクションで画面表示の優先順位を変更した場合は，「画面マスクの設定(AH=0CH)」ファンクションを用いて画面マスクを再設定する必要がある．

ただし，GS以外の機種では常に制御なしにしなければならない(画面マスク機能が使用されていない場合はこの限りではない)．

マスク制御ありの場合，優先順位に応じて画面マスクも変更される．GS以外の機種では，マスク制御をもたないため意味をもたない．

注　意

すべての画面を表示禁止にすることはできない.

「画面モードの設定(AH=02H)」で確保できたフレーム以外は使用できない.

優先状態のパラメータに指定なし(000)が設定されると，以下の優先状態は無効となる.

$ASKCOMP	AH=08H	
画面優先順位の取得		

入　力

AH = 08H

出　力

Carry = 実行ステータス
- 0 ： 正常設定
- 1 ： 「画面モードの設定(AH=02H)」による画面モードが設定されていない

AL = テキストスーパー優先
- 0 ： テキストはグラフィックスと一体
- 1 ： テキスト最優先(GSのみ)

BH ＝ 第1画面/第2画面状態

D7	D6	D5	D4	D3	D2	D1	D0
0		F		0		S	

第2画面指定 (D2–D0: S)

000：指定なし
001：カレント画面
010：画面0（GSのみ）
011：画面1（GSのみ）
100：画面2（GSのみ）
101：画面3（GSのみ）
110：ビデオ画面

第2画面指定 (D6–D4: F)

000：指定なし（GSのみ）
001：カレント画面
010：画面0
011：画面1（GSもしくは
　　　9821 400ラインモード時）
100：画面2（GSのみ）
101：画面3（GSのみ）
110：ビデオ画面

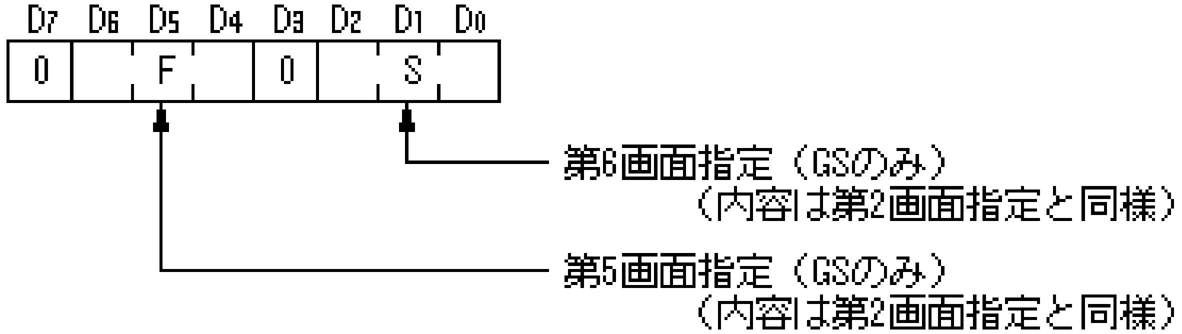
BL ＝ 第3画面/第4画面状態

D7	D6	D5	D4	D3	D2	D1	D0
0		T		0		F	

第4画面指定（GSのみ） (D2–D0: F)
　　（内容は第2画面指定と同様）

第3画面指定（GSのみ） (D6–D4: T)
　　（内容は第2画面指定と同様）

CH = 第5画面/第6画面状態



機　能

現在の画面合成の優先順位を取得する.

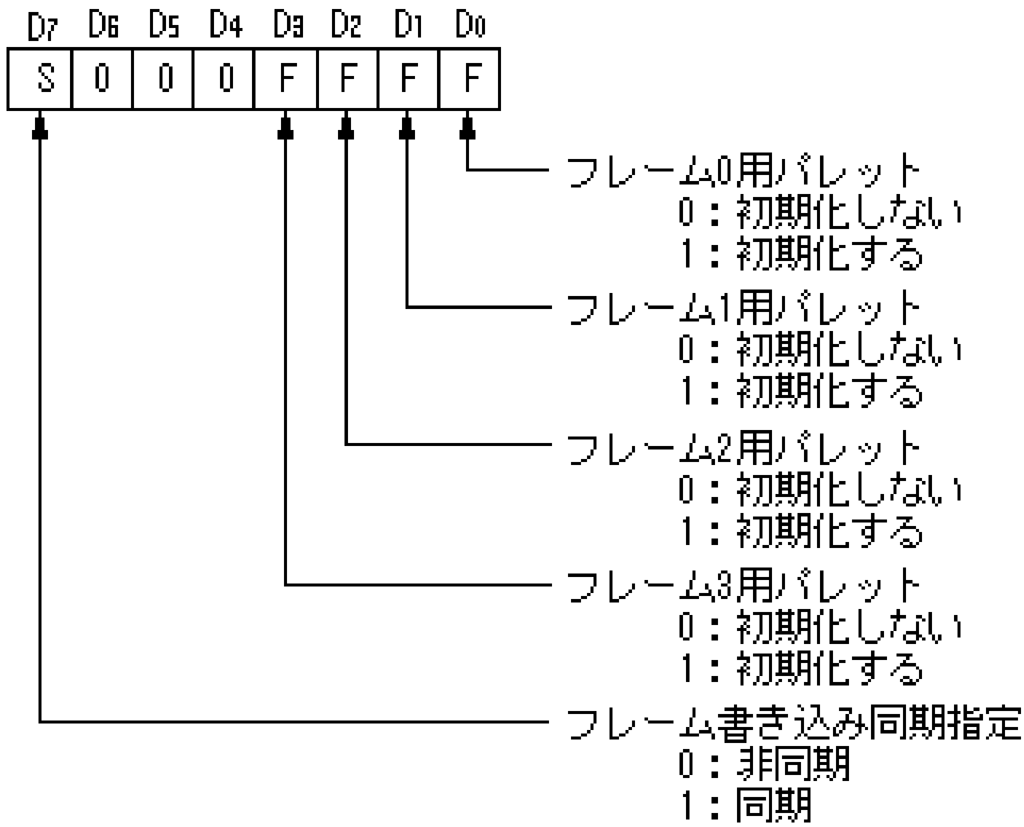
$INIPAL	AH=09H	

パレットの初期化

入　力

AH = 09H

AL = パレット指定



出　力

Carry = 初期化ステータス

0 ： 初期化完了

1 ： 「画面モードの設定(AH=02H)」による画面モードの設定が行われていない

機　能

各フレームに対するカラーパレットを初期化する．

8ビット画面モードになっているフレームに対してのみ初期化を行い，それ以外のパレットには何もしない．GS以外は1セットのパレットのみ初期化を行う．

初期化の値は次のようになる．

●16ビットパレット時(GSのみ)

カラーコード

D7	D6	D5	D4	D3	D2	D1	D0
B7	B6	B5	B4	B3	B2	B1	B0

→青レベル

D7	D6	D5	D4	D3	D2	D1	D0
B1	B0	B1	B0	B1			

→赤レベル

D7	D6	D5	D4	D3	D2	D1	D0
B4	B3	B2	B4	B3			

→緑レベル

D7	D6	D5	D4	D3	D2	D1	D0
B7	B6	B5	B7	B6	B5		

●24ビットパレット時

カラーコード

D7	D6	D5	D4	D3	D2	D1	D0
B7	B6	B5	B4	B3	B2	B1	B0

→青レベル

D7	D6	D5	D4	D3	D2	D1	D0
B1	B0	B1	B0	B1	B0	B1	B0

→赤レベル

D7	D6	D5	D4	D3	D2	D1	D0
B4	B3	B2	B4	B3	B2	B4	B3

→緑レベル

D7	D6	D5	D4	D3	D2	D1	D0
B7	B6	B5	B7	B6	B5	B7	B6

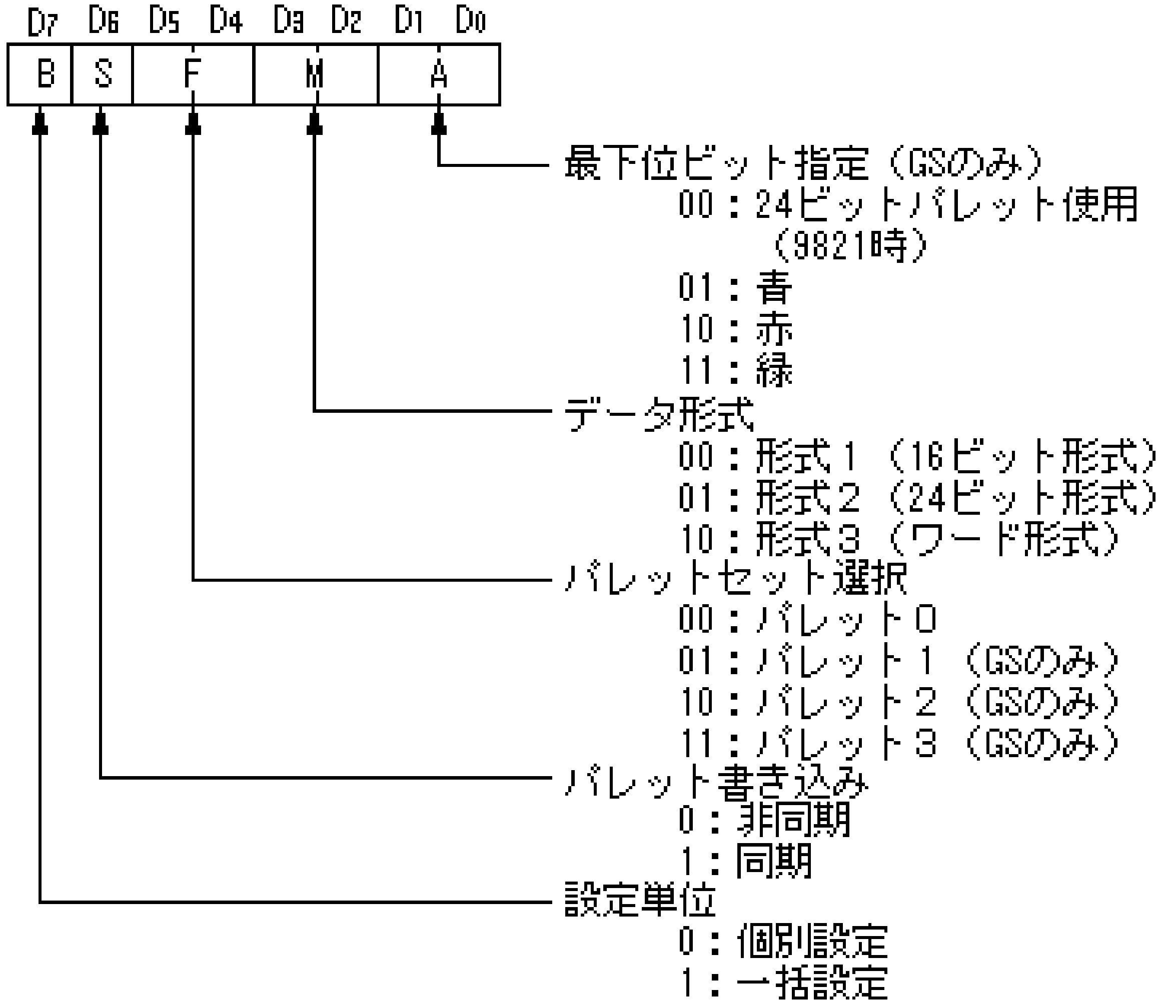
$SETPAL	AH=0AH	

パレットの設定

入　力

AH = 0AH

ALフレーム選択/モード選択



BH = パレット番号(個別設定の場合)
BL = 青レベル(個別設定，形式3以外の場合)
CH = 赤レベル(個別設定，形式3以外の場合)
CL = 緑レベル(個別設定，形式3以外の場合)
CX = パレット値(個別設定，形式3の場合)
ES:BX = データテーブル格納アドレス(一括設定の場合)

出　力

Carry = 初期化ステータス
0 ： 設定完了
1 ： 「画面モードの設定(AH=02H)」 による画面モードの設定が行われていない
パラメータが不適切

機　能

以下の各形式に従って，パレット値を設定する.

●形式1（16ビットパレットのデータ形式）

BRGを各1バイトずつ使用し，以下のようなデータ形式とする.

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
G5	G4	G3	G2	G1	AT(*1)	R5	R4	R3	R2	R1	B5	B4	B3	B2	B1

G（緑）　R（赤）　B（青）

（*1）（GSのみ）有効

最下位ビット指定が　青の場合：B0
赤の場合：R0
緑の場合：G0

・16ビットパレットを参照している場合(GSのみ)

入力値

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
G5	G4	G3	G2	G1	AT(*1)	R5	R4	R3	R2	R1	B5	B4	B3	B2	B1

↓

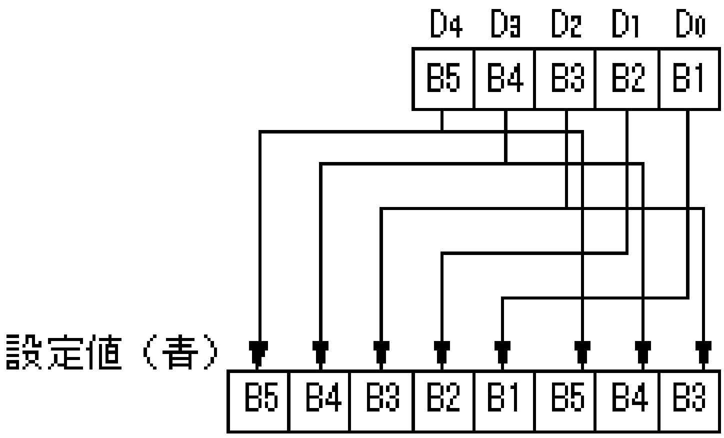
設定値

G5	G4	G3	G2	G1	AT(*1)	R5	R4	R3	R2	R1	B5	B4	B3	B2	B1

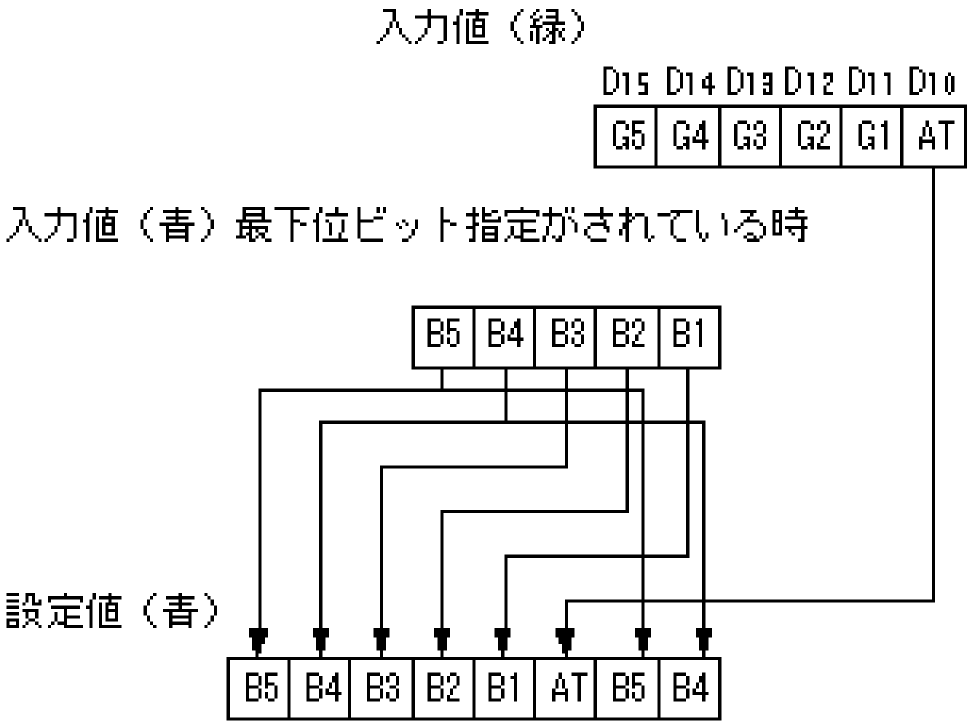
G（緑）　R（赤）　B（青）

・フレームが24ビットパレットを参照している場合

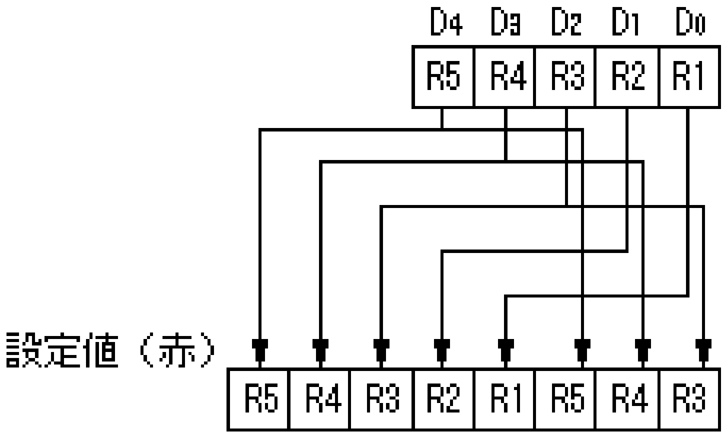
入力値（青）最下位ビット指定がされていない時



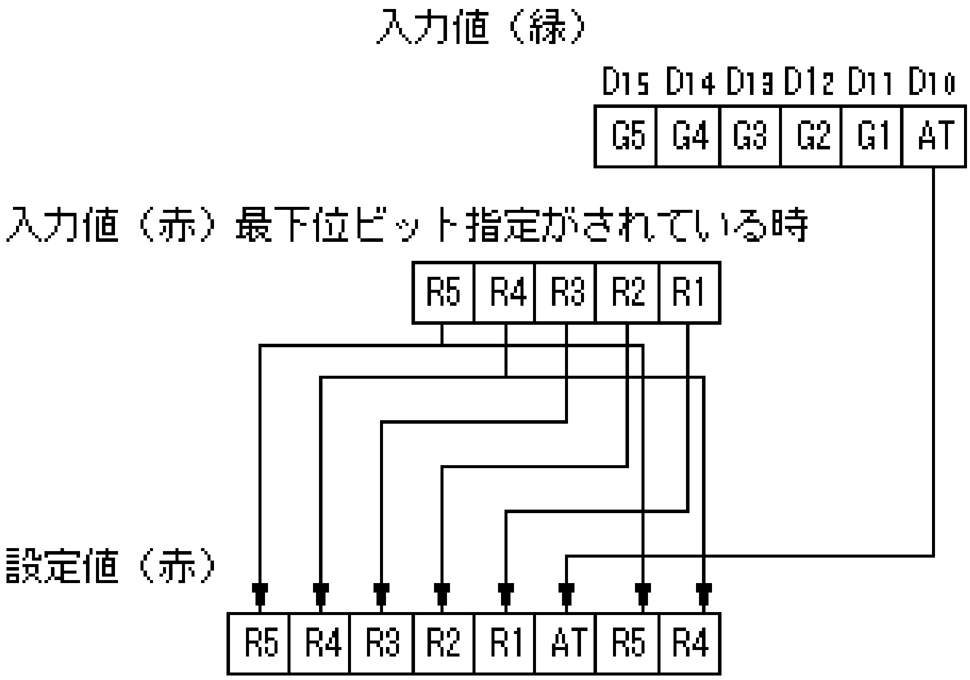
入力値（青）最下位ビット指定がされている時



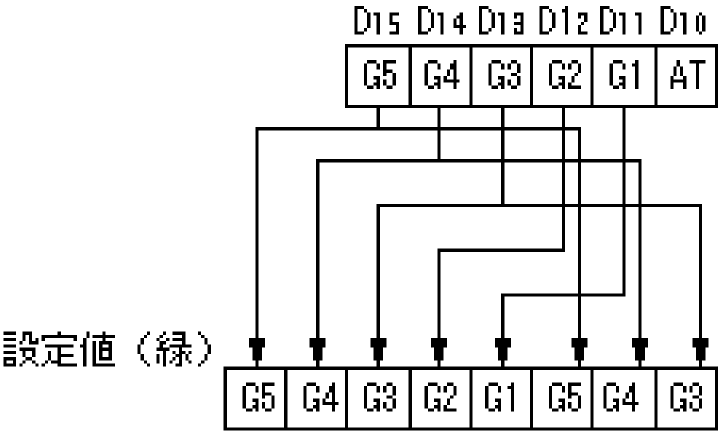
入力値（赤）最下位ビット指定がされていない時



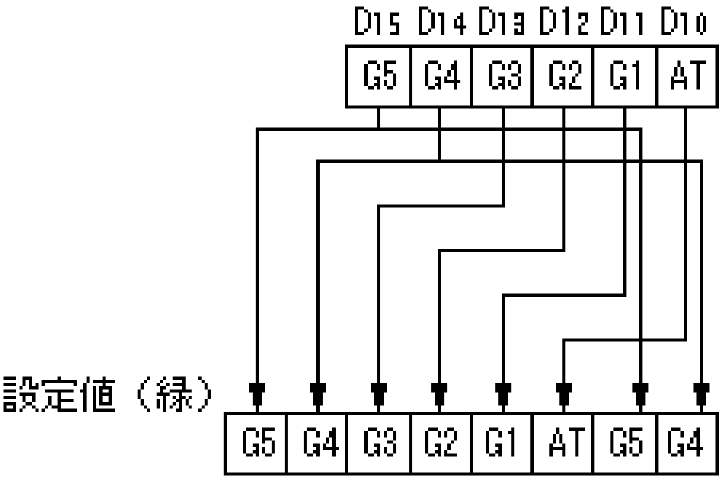
入力値（赤）最下位ビット指定がされている時



入力値（緑）最下位ビット指定がされていない時



入力値（緑）最下位ビット指定がされていない時



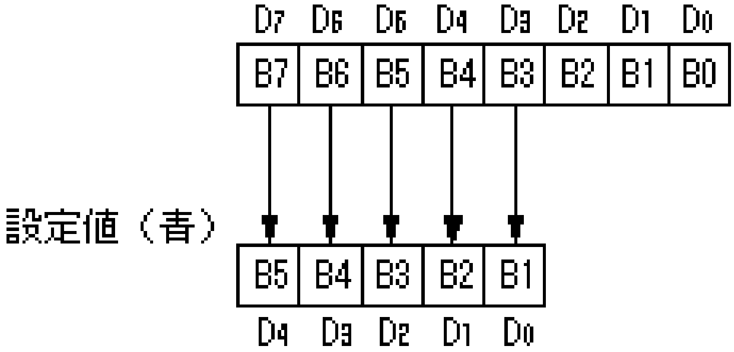
●形式2（24ビットパレットのデータ形式）

BRGを各1バイトずつ使用し，以下のようなデータ形式とする．

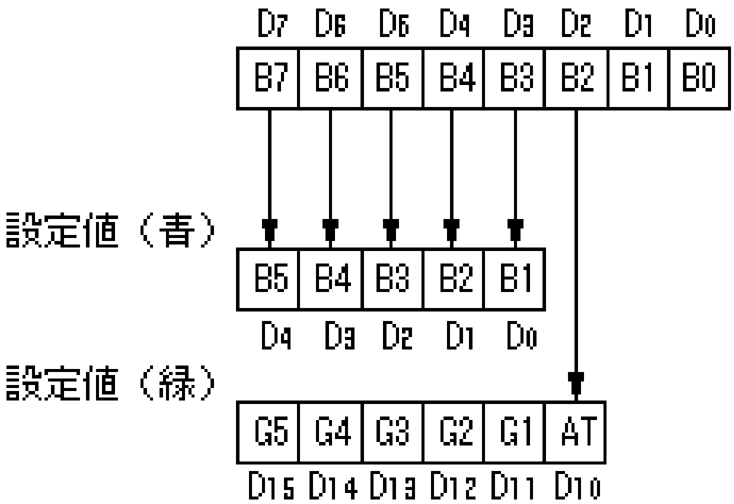
	D7	D6	D5	D4	D3	D2	D1	D0
B（青）	B7	B6	B5	B4	B3	B2	B1	B0
R（赤）	R7	R6	R5	R4	R3	R2	R1	R0
G（緑）	G7	G6	G5	G4	G3	G2	G1	G0

・フレームが16ビットパレットを参照している場合(GSのみ)

入力値（青）最下位ビットが指定されていない時

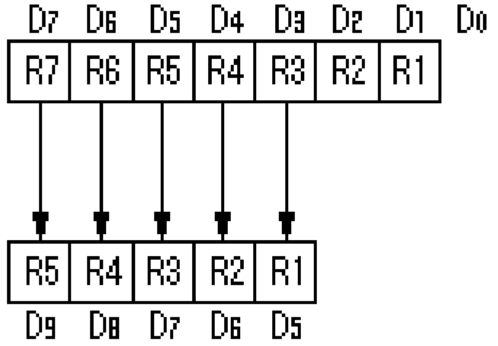


入力値（青）最下位ビットが指定されている時



入力値（赤）最下位ビットが指定されていない時

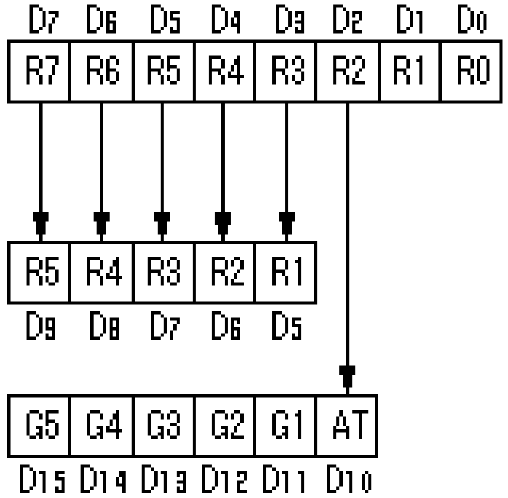
設定値（赤）



入力値（赤）最下位ビットが指定されている時

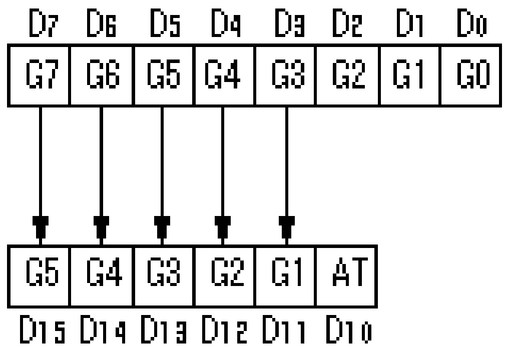
設定値（赤）

設定値（緑）

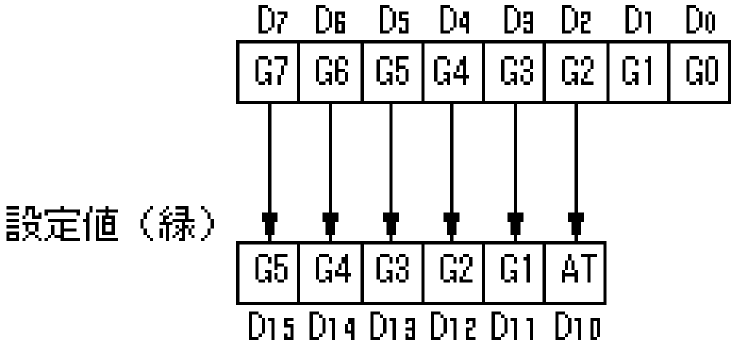


入力値（緑）最下位ビットが指定されていない時

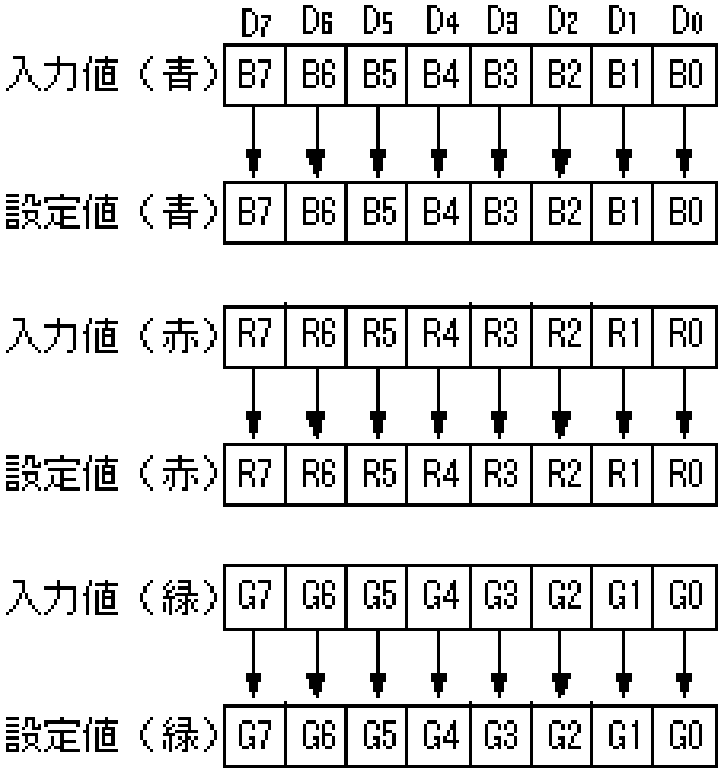
設定値（緑）



入力値（緑）最下位ビットが指定されている時



・フレームが24ビットパレットを参照している場合

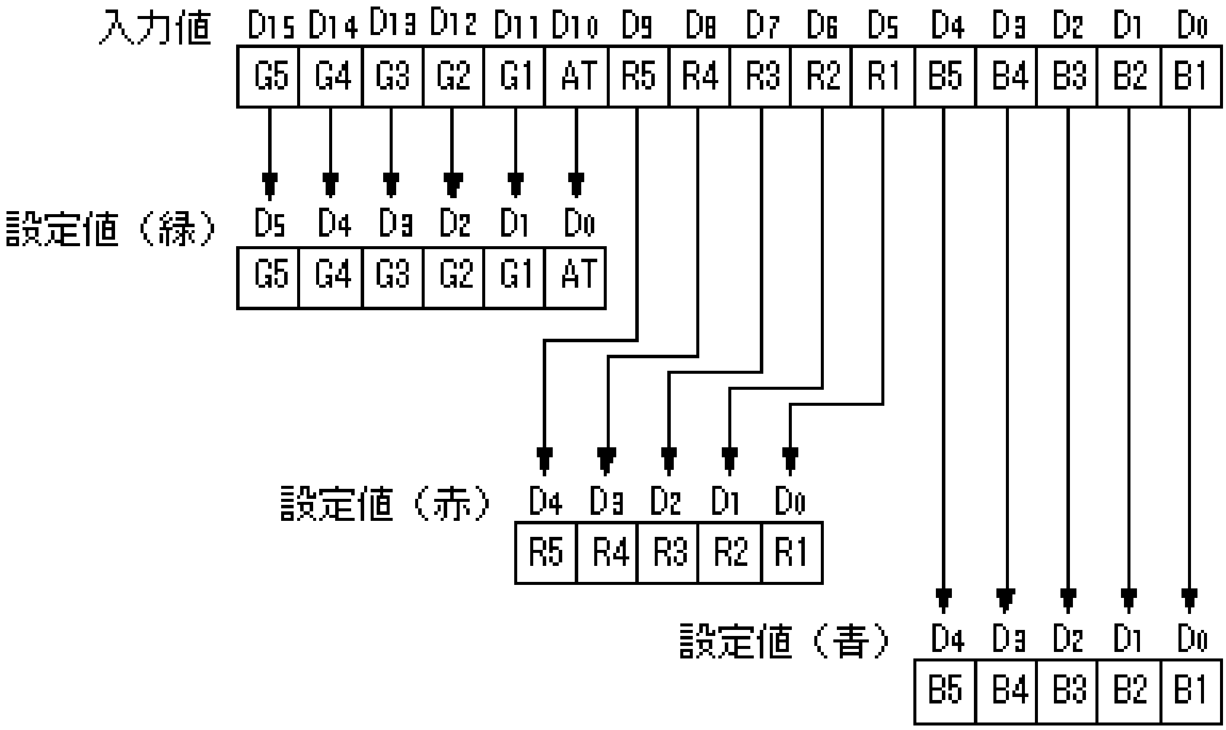


●形式3（ワード形式）

BRGを1ワードにまとめ，以下のようなデータ形式とする．

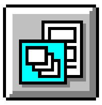
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
G5	G4	G3	G2	G1	AT	R5	R4	R3	R2	R1	B5	B4	B3	B2	B1

・フレームが16ビットパレットを参照している場合(GSのみ)

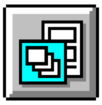


・フレームが24ビットパレットを参照している場合

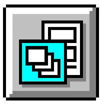
▼24ビットパレット参照 LSBが青（形式3）



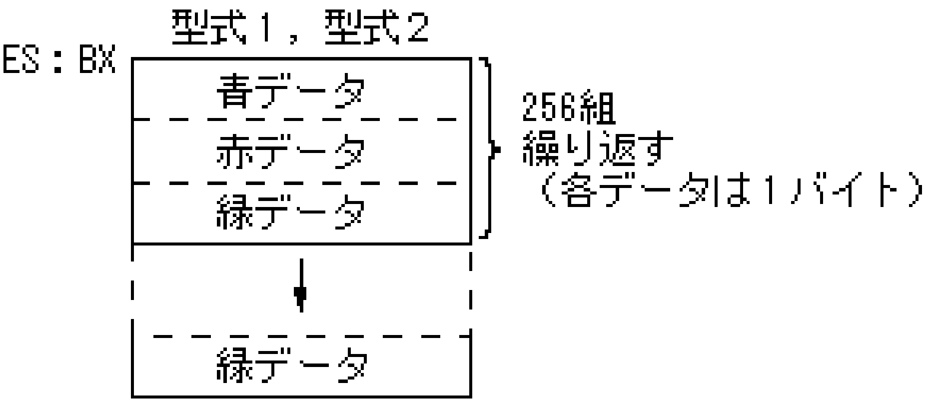
▼24ビットパレット参照 LSBが赤（形式3）

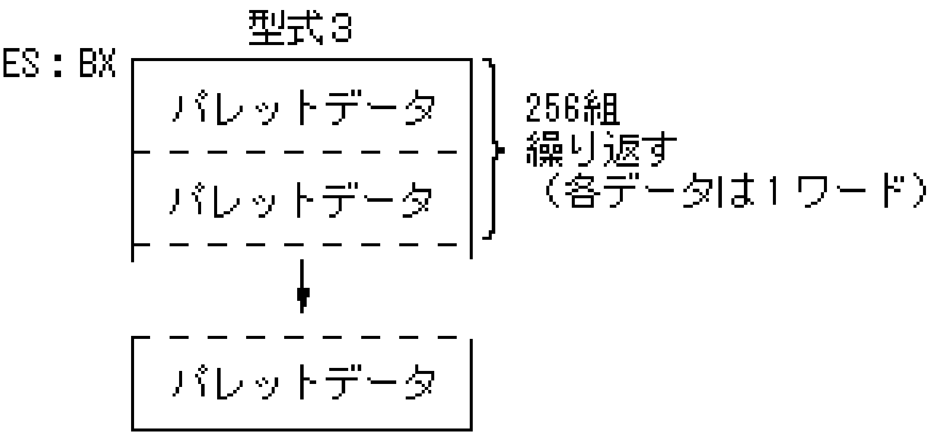


▼24ビットパレット参照 LSBが緑（形式3）



各データテーブルの格納形式は以下のとおり．





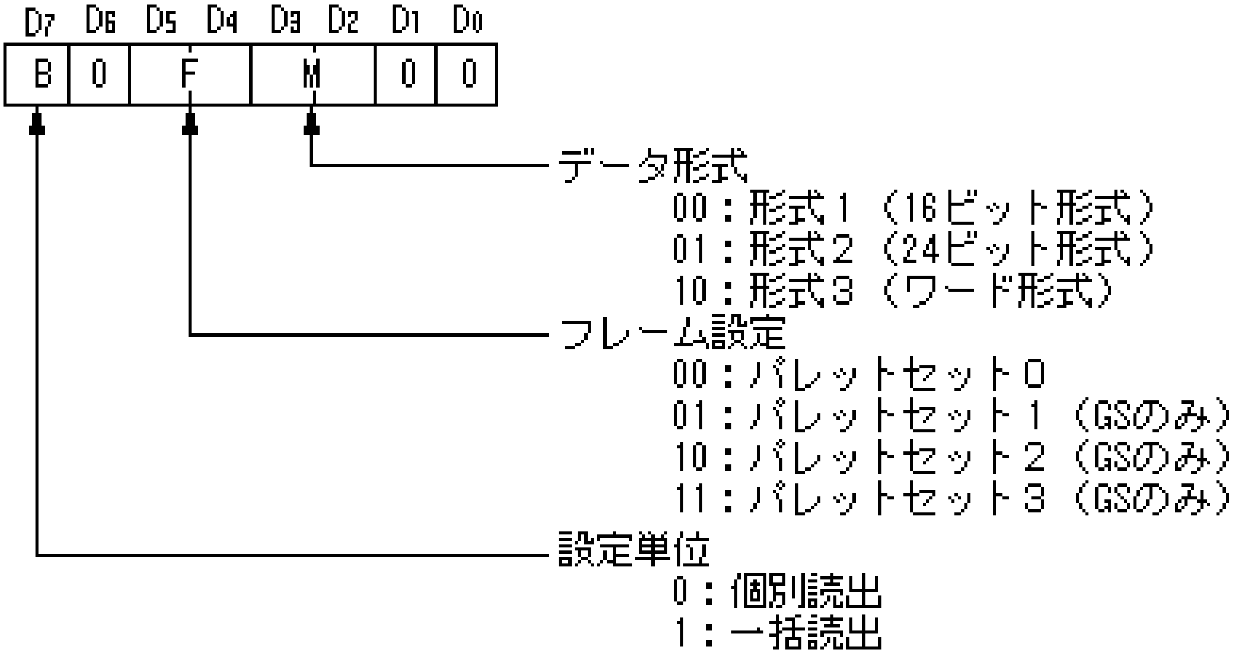
$ASKPAL	AH=0BH	

パレットの取得

入　力

AH = 0BH

AL = フレーム選択/モード選択



BH = パレット番号(個別読出の場合)

ES:BX = データテーブル格納アドレス(一括読出の場合)

出　力

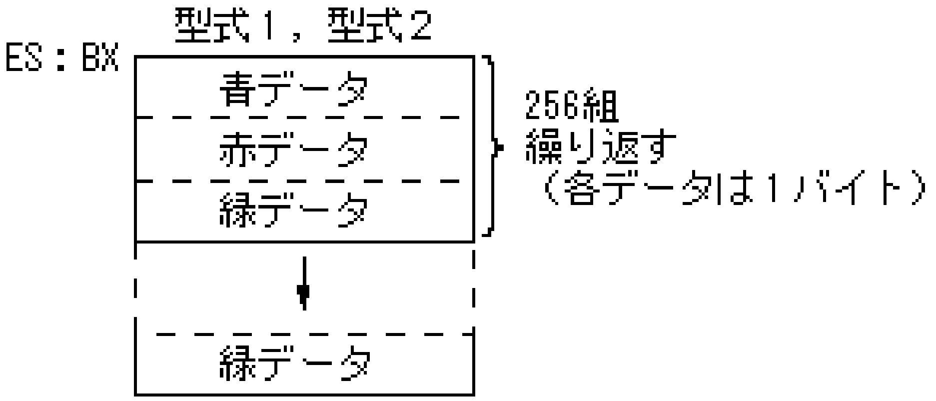
Carry = 実行ステータス

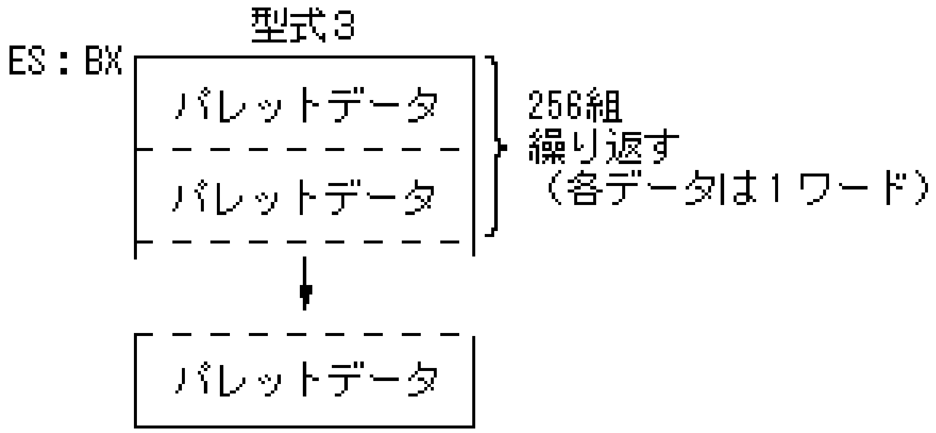
0 ： 読出完了

1 ： 「画面モードの設定(AH=02H)」 による画面モードの設定が行われていない

BL = 青レベル(個別読出，形式3以外の場合)
CH = 赤レベル(個別読出，形式3以外の場合)
CL = 緑レベル(個別読出，形式3以外の場合)
CX = パレット値(個別読出，形式3の場合)
ES:BX = データテーブル格納アドレス(一括読出の場合)

各データテーブルの格納形式は以下のとおり．





機　能

現在のパレットを取得する．

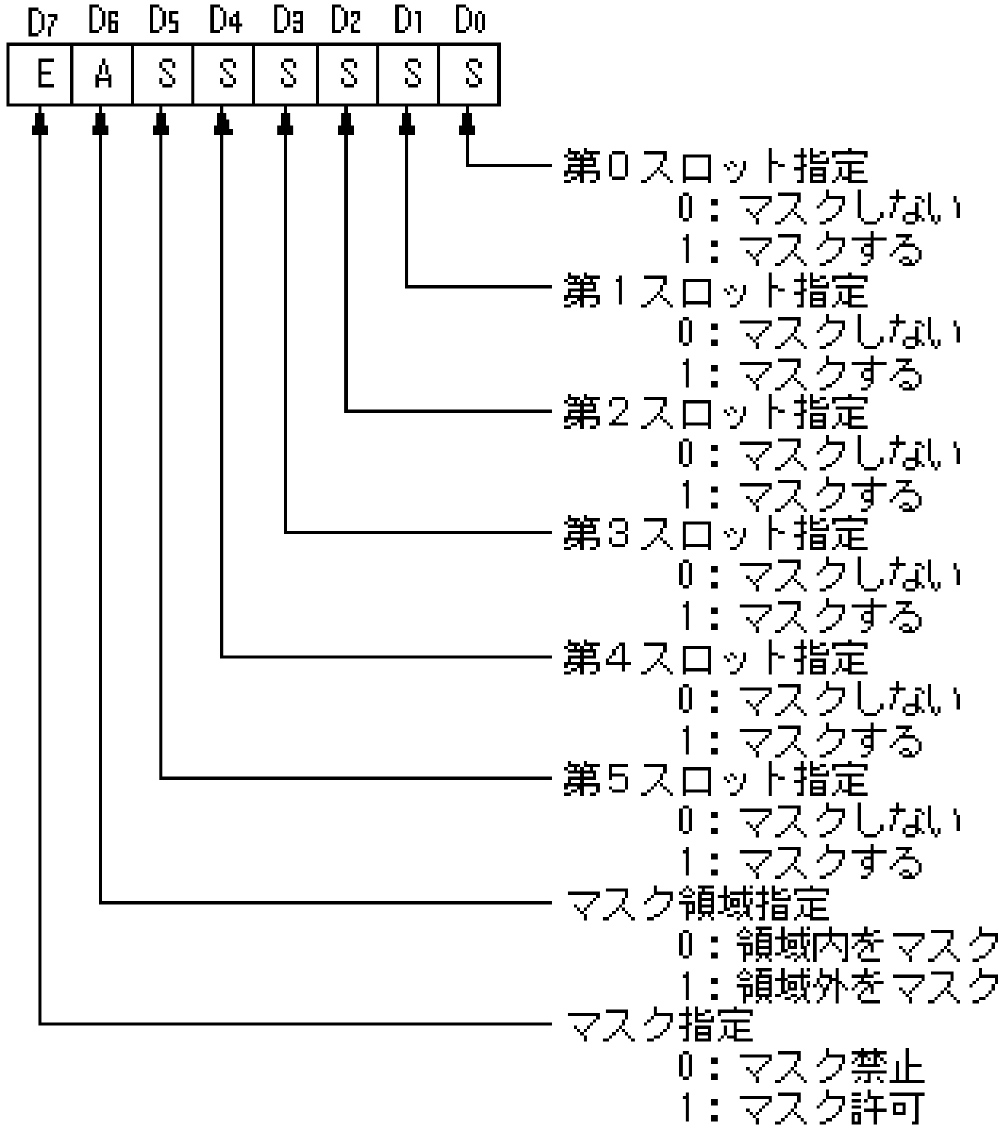
$SETMASK	AH=0CH	GS

画面マスクの設定

入　力

AH ＝ 0CH

AL ＝ マスクスロット指定



BX = マスク開始水平位置(マスクボックスの左端)
CX = マスク終了水平位置(マスクボックスの右端)
DH = マスク開始垂直位置(マスクボックスの上端)
DL = マスク終了垂直位置(マスクボックスの下端)

出　力

Carry = 設定ステータス
　　0 ： 設定完了
　　1 ： エラー
AH = エラーステータス
　　1 ： 「画面モードの設定(AH=02H)」による画面モードの設定が行われていない
　　2 ： 入力パラメータ異常

機　能

画面マスクを設定する.

$ASKMASK	AH=0DH	GS
画面マスクの取得		

入　力

AH = 0DH

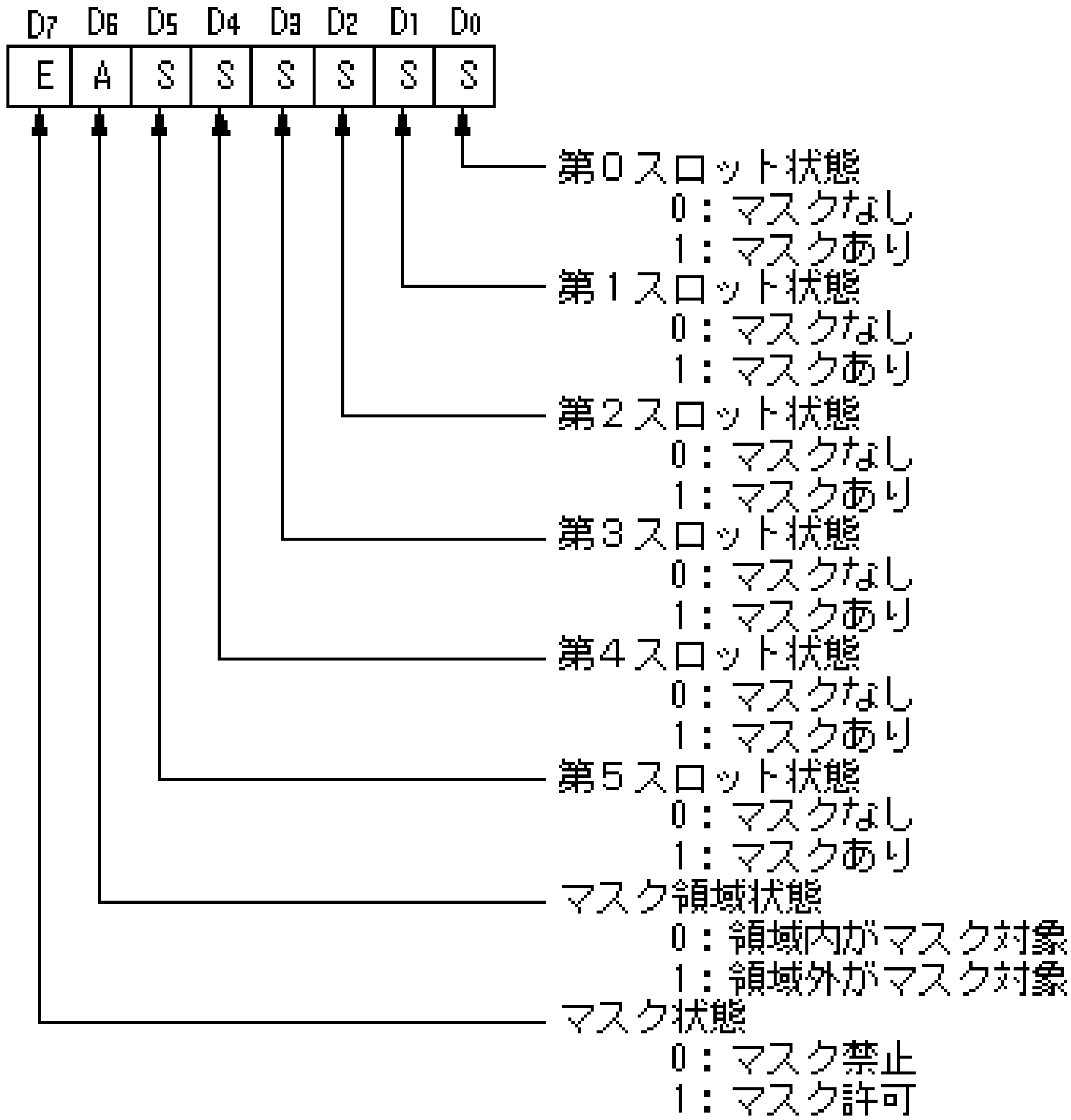
出　力

Carry = 獲得ステータス

0 ： 獲得完了

1 ： 「画面モードの設定(AH=02H)」による画面モードの設定が行われていない

AL = マスクスロット状態



BX = マスク開始水平位置(マスクボックスの左端)

CX = マスク終了水平位置(マスクボックスの右端)

DH = マスク開始垂直位置(マスクボックスの上端)

DL = マスク終了垂直位置(マスクボックスの下端)

機　能

現在の画面マスクを取得する.

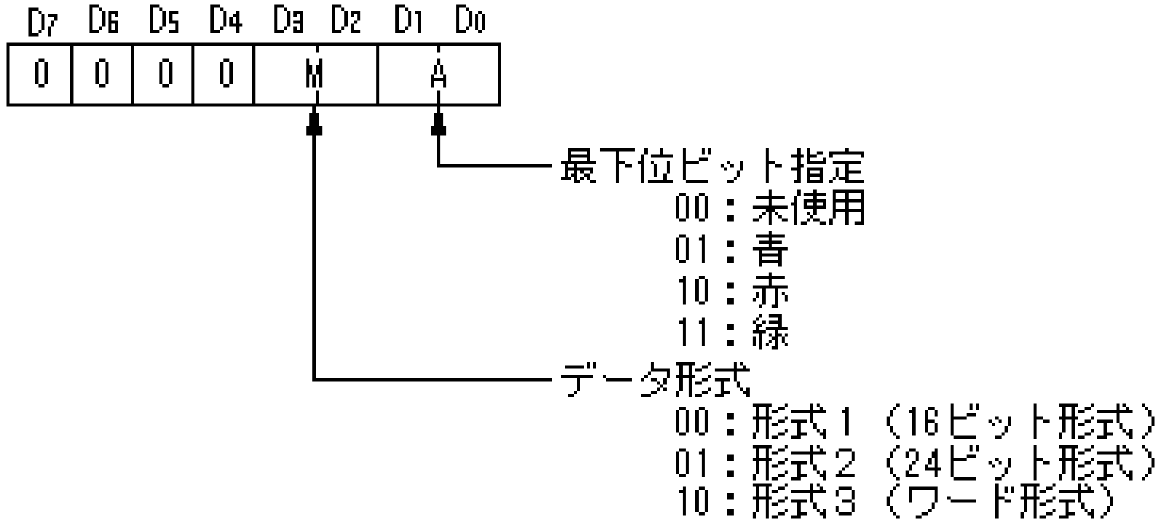
$SETBACK	AH=0EH	GS

背景色の設定

入　力

AH = 0EH

AL = モード選択



BL = 青レベル(形式3以外の場合)

CH = 赤レベル(形式3以外の場合)

CL = 緑レベル(形式3以外の場合)

CX = ワード値(形式3の場合)

出　力

Carry = 初期化ステータス

0 ： 設定完了

1 ： データ形式が不適切

機　能

背景色を指定された色レベルにより設定する.
色レベルの形式は「パレットの設定(AH=0AH)」と同様.

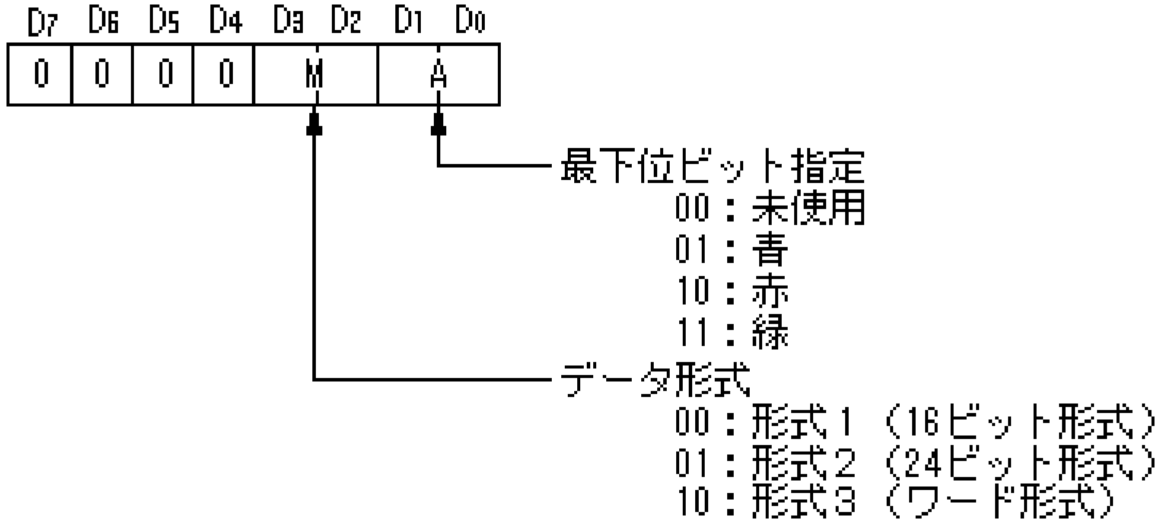
$ASKBACK	AH=0FH	GS

背景色の取得

入　力

AH = 0FH

AL = モード選択



出　力

Carry = 獲得ステータス

　　0 ： 獲得完了

　　1 ： データ形式が不適当

BL = 青レベル(形式3以外の場合)

CH = 赤レベル(形式3以外の場合)

CL = 緑レベル(形式3以外の場合)

CX = ワード値(形式3の場合)

機　能

現在の背景色を取得する.

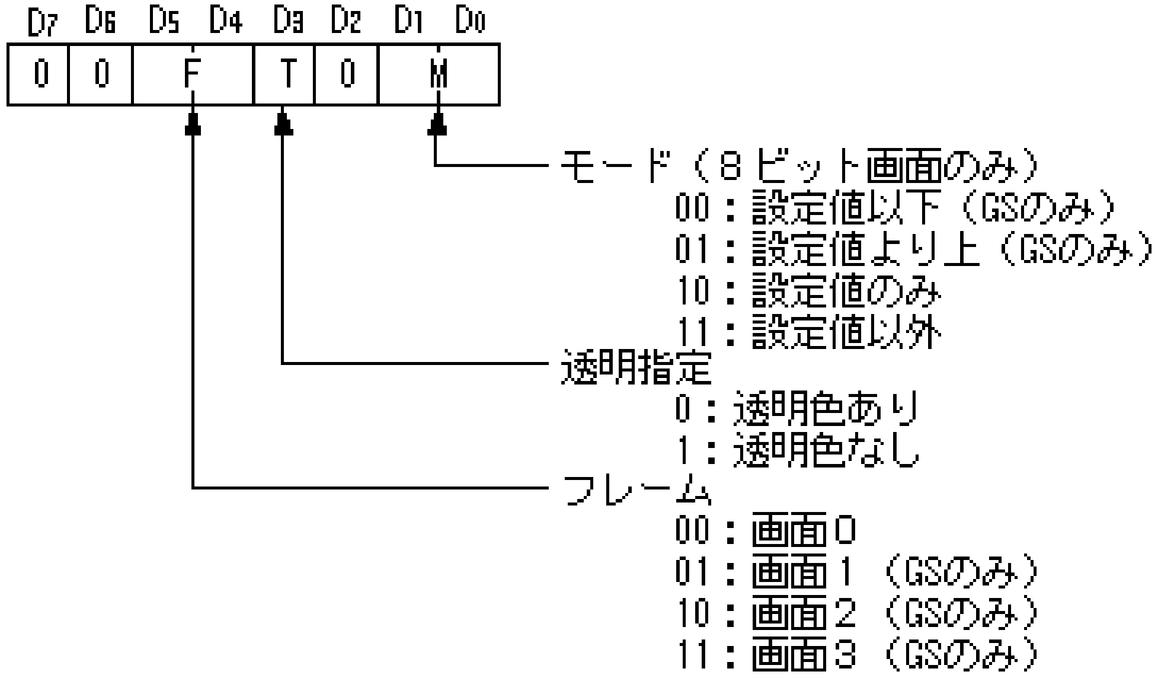
$SETTHRU	AH=10H	

透明色の設定

入　力

AH ＝ 10H

AL ＝ フレーム選択/モード設定



「透明色なし」の場合，以下のパラメータは不要.

BH ＝ カラーコード番号(設定値)(8ビット画面の場合)
　　　青アッパーレベル(8ビット画面以外の場合)(GSのみ)

BL ＝ 青ロワーレベル(8ビット画面以外の場合)(GSのみ)

CH ＝ 赤アッパーレベル(8ビット画面以外の場合)(GSのみ)

CL ＝ 赤ロワーレベル(8ビット画面以外の場合)(GSのみ)

DH ＝ 緑アッパーレベル(8ビット画面以外の場合)(GSのみ)

DL ＝ 緑ロワーレベル(8ビット画面以外の場合)(GSのみ)

出　力

Carry = 設定ステータス
- 0 ： 設定完了
- 1 ： エラー

AH = エラーステータス
- 1 ： 「画面モードの設定(AH=02H)」 による画面モードの設定が行われていない
- 2 ： フレーム未定義
- 3 ： ビデオボード未実装

機　能

ALで指定されたフレームに対し透明色を設定する.

透明色を設定した場合，裏のビデオ画面を表示することができる.

8ビット画面の場合は，BHで指定された設定値(カラーコード番号)を基準に透明色が設定される.

8ビット画面以外の場合は，BX，CX，DXで指定された，各色成分におけるレベルの範囲で透明色が設定される(GSのみ).

$ASKTHRU	AH=11H	
透明色の取得		

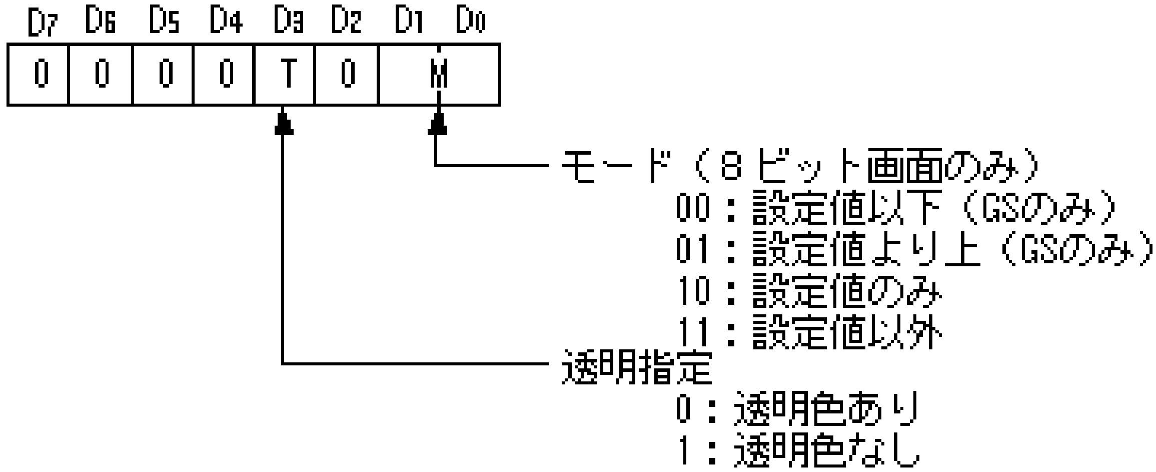
入　力

AH = 11H
AL = フレーム選択
　　00H ： 画面0
　　10H ： 画面1(GSのみ)
　　20H ： 画面2(GSのみ)
　　30H ： 画面3(GSのみ)

出　力

Carry = 獲得ステータス
　　0 ： 獲得完了
　　1 ： エラー
AH = エラーステータス (Carry=1の場合)
　　1 ： 「画面モードの設定(AH=02H)」 による画面モードの設定が行われていない
　　2 ： 透明範囲不整合
　　3 ： ビデオボード未実装

AL = モード状態(Carry = 0の場合)



BH = カラーコード番号(8ビット画面の場合)
　　　青アッパーレベル(8ビット画面以外の場合)(GSのみ)
BL = 青ロワーレベル(8ビット画面以外の場合)(GSのみ)
CH = 赤アッパーレベル(8ビット画面以外の場合)(GSのみ)
CL = 赤ロワーレベル(8ビット画面以外の場合)(GSのみ)
DH = 緑アッパーレベル(8ビット画面以外の場合)(GSのみ)
DL = 緑ロワーレベル(8ビット画面以外の場合)(GSのみ)

機　能

ビデオ画面に対する透明色の設定を取得する.

$QUANTUM	AH=12H	GS

表示色量子化状態の設定

入　力

AH = 12H
BL = 青量子化レベル(0〜7 ： 落とすビット数)
CH = 赤量子化レベル(0〜7 ： 落とすビット数)
CL = 緑量子化レベル(0〜7 ： 落とすビット数)

出　力

Carry = エラーステータス
　0 ： 正常終了
　1 ： パラメータ異常

機　能

各色の落とすビット数に合わせて量子化設定レジスタを設定する.

$ASKQUANT	AH=13H	GS
表示色量子化状態の取得		

入　力

AH = 13H

出　力

BL = 青量子化レベル(0～7 ： 落とすビット数)
CH = 赤量子化レベル(0～7 ： 落とすビット数)
CL = 緑量子化レベル(0～7 ： 落とすビット数)

機　能

現在の表示色の量子化状態を取得する.

$EFFECT1	AH=14H	GS

特殊効果の設定[表示優先順位依存]

入　力

AH = 14H
AL = 特殊効果指定
0xH ： 特殊効果禁止(GSのみ)
3xH ： 透明合成(通常合成)
5xH ： 加算合成(GSのみ)
6xH ： フィルタ処理(GSのみ)
7xH ： 半透明処理
40H ： 全ビットOFF(表示は黒になる)(GSのみ)
41H ： not(A or B)(GSのみ)
42H ： not(A) and B(GSのみ)
43H ： not(A)(GSのみ)
44H ： A and not(B)(GSのみ)
45H ： not(B)(GSのみ)
46H ： A xor B(GSのみ)
47H ： not(A and B)(GSのみ)
48H ： A and B(GSのみ)
49H ： not(A xor B)(GSのみ)
4AH ： B(GSのみ)
4BH ： not(A) or B(GSのみ)
4CH ： A(GSのみ)
4DH ： not(B) or A(GSのみ)
4EH ： A or B(GSのみ)
4FH ： 全ビットON(表示は白になる)(GSのみ)

出　力

Carry = 実行ステータス
　　0 ： 正常設定
　　1 ： エラー
AH = エラーステータス
　　1 ： パラメータ異常
　　2 ： 特殊効果設定不可

機　能

最優先画面と第2優先画面間で演算を行い特殊効果を出す．

上位4ビットで効果モード，下位4ビットで論理演算を指定する．論理演算は効果モードが4の場合だけ機能し，その他の効果モードでは無視される．

効果モードは3～7までが有効で，0の場合は特殊効果が禁止される．上記以外の値が指定された場合は，パラメータエラーとしてCarryに1を返す．

この「特殊効果の設定[表示優先順位依存](AH=14H)」は，特殊効果設定とは排他的に機能し，最後に使用されたファンクションに従って特殊効果機能が設定される．

特殊効果はカラーパレットが24ビットかつ以下の条件では機能しない(Carry=1，AL=2となる)．

・フレーム0が1024×256 8ビット/ピクセルの場合
・フレーム3が1024×256 8ビット/ピクセルの場合
（フレーム0が24ビット/ピクセルの場合を除く）

参　考

　フェード制御を行う場合には，あらかじめ「特殊効果の設定［表示優先順位依存］（AH=14H）」，「特殊効果の設定［優先画面指定］（AH=15H）」で特殊効果を設定しなければならない．（GSのみ）
　通常表示にフェード制御を加える場合は，「特殊効果の設定［表示優先順位依存］（AH=14H）」，「特殊効果の設定［優先画面指定］（AH=15H）」の特殊効果指定として3xHを設定する．（GSのみ）

特殊効果

透明合成
　　最優先画面の黒の部分が第2優先画面になる
加算合成
　　最優先画面と第2優先画面を加算する（GSのみ）
フィルタ処理
　　第2優先画面を最優先画面のレベルに対応させる
　　第2優先画面-NOT（最優先画面）を行う（GSのみ）
半透明処理
　　最優先画面と第2優先画面の平均をとる
論理演算合成処理
　　最優先画面と第2優先画面間で論理演算を行う（GSのみ）

※GS以外の機種では，半透明処理/透明合成のみ使用可能

$EFFECT2	AH=15H	
特殊効果の設定［優先画面指定］		

入　力

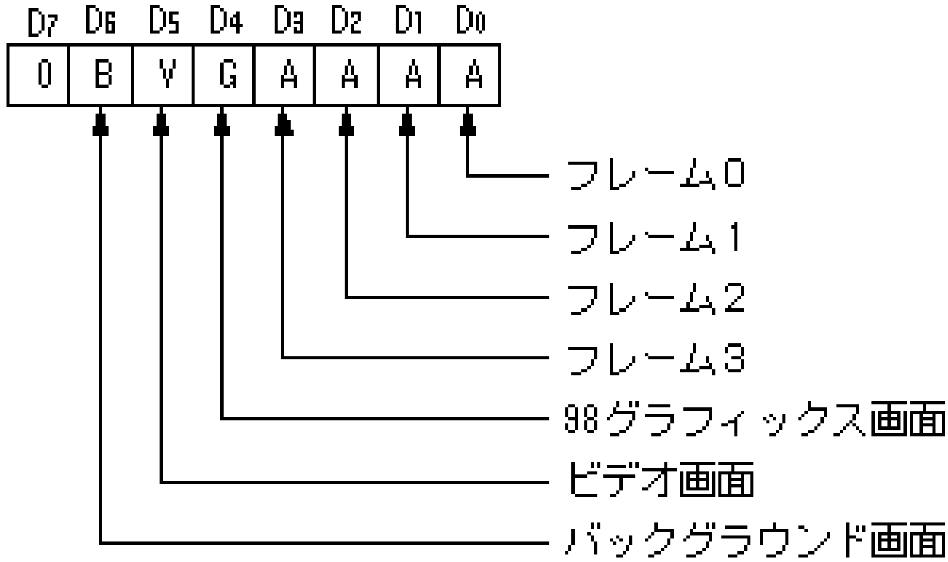
AH = 15H

AL = 特殊効果指定

0xH ： 特殊効果禁止
3xH ： 透明合成(通常合成)
5xH ： 加算合成
6xH ： フィルタ処理
7xH ： 半透明処理
40H ： 全ビットOFF(表示は黒になる)
41H ： not(A or B)
42H ： not(A) and B
43H ： not(A)
44H ： A and not(B)
45H ： not(B)
46H ： A xor B
47H ： not(A and B)
48H ： A and B
49H ： not(A xor B)
4AH ： B
4BH ： not(A) or B
4CH ： A
4DH ： not(B) or A
4EH ： A or B
4FH ： 全ビットON(表示は白になる)

BH ＝ 最優先画面指定
　　00H ： フレーム0
　　01H ： フレーム1
　　02H ： フレーム2
　　03H ： フレーム3

BL ＝ 次優先画面指定(使用する画面に対応するビットを1にする)



CH ＝ 効果範囲色モード指定
　　(BHで指定されたフレームが8ビット/ピクセルのみ有効)
　　00H ： CLでの指定色以下
　　01H ： CLでの指定色より上
　　02H ： CLでの指定色のみ
　　03H ： CLでの指定色以外

CL ＝ 効果範囲色指定
　　(BHで指定されたフレームが8ビット/ピクセルのみ有効)

出　力

Carry = 実行ステータス
　　0 ： 正常設定
　　1 ： エラー
AH = エラーステータス
　　1 ： パラメータ異常
　　2 ： 特殊効果設定不可
　　3 ： 指定されたフレームが未定義
　　4 ： 画面合成の優先順位不適合

機　能

最優先画面と次優先画面間で演算を行い特殊効果を出す.

上位4ビットで効果モード, 下位4ビットで論理演算を指定する. 論理演算は効果モードが4の場合だけ機能し, その他の効果モードでは無視される. 効果モードは3～7までが有効で, 0の場合は特殊効果が禁止される.

上記以外の値が指定された場合は, パラメータエラーとしてCarryに1を返す.

・特殊効果の処理内容
　① 優先画面から指定されたカラーコード範囲の抽出
　② ①で抽出された画面と次優先画面の演算

この「特殊効果の設定[優先画面指定](AH=15H)」は, 「特殊効果の設定[表示優先順位依存](AH=14H)」とは排他的に機能し, 最後に使用されたファンクションに従って特殊効果機能が設定される.

特殊効果はカラーパレットが24ビットかつ以下の条件では機能しない(Carry=1，AL=2となる).

・フレーム0が1024×256 8ビット/ピクセルの場合
・フレーム3が1024×256 8ビット/ピクセルの場合
（フレーム0が24ビット/ピクセルの場合を除く）

最優先画面として指定された画面よりも，次優先画面として指定された画面の方が「画面優先順位の設定(AH=07H)」により前に合成されている場合は，特殊効果機能が使用できない(Carry=1，AL=3となる).

最優先画面が8ビット/ピクセルの場合は，カラーコードにより範囲指定された領域のみが特殊効果の対象となる.

特殊効果の内容に関しては，「特殊効果の設定[表示優先順位依存](AH=14H)」を参照すること.

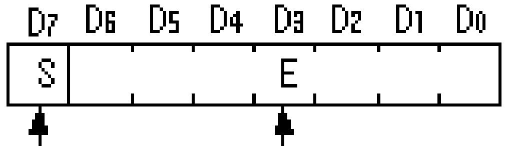
$ASKEFFECT	AH=16H	
特殊効果状態の取得		

入　力

AH = 16H

出　力

AL ＝ 特殊効果状態



特殊効果モード
- 0xH：特殊効果禁止（GSのみ）
- 3xH：透明合成（通常合成）
- 5xH：加算合成（GSのみ）
- 6xH：フィルタ処理（GSのみ）
- 7xH：半透明処理
- 40H：全ビットOFF（表示は黒）（GSのみ）
- 41H：not(A or B)（GSのみ）
- 42H：not(A) and B（GSのみ）
- 43H：not(A)（GSのみ）
- 44H：A and not(B)（GSのみ）
- 45H：not(B)（GSのみ）
- 46H：A xor B（GSのみ）
- 47H：not(A and B)（GSのみ）
- 48H：A and B（GSのみ）
- 49H：not(A xor B)（GSのみ）
- 4AH：B（GSのみ）
- 4BH：not(A) or B（GSのみ）
- 4CH：A（GSのみ）
- 4DH：not(A) or A（GSのみ）
- 4EH：A or B（GSのみ）
- 4FH：全ビットON（表示は白）（GSのみ）

優先画面順位
- 0 ：優先画面指定特殊効果の設定
  ［優先画面指定］（AH＝15H）
- 1 ：表示順位に依存（特殊効果の設定）
  ［優先画面順位依存］（AH＝14H）
  （BX,CXの値は意味を持たない）

BH = 最優先画面

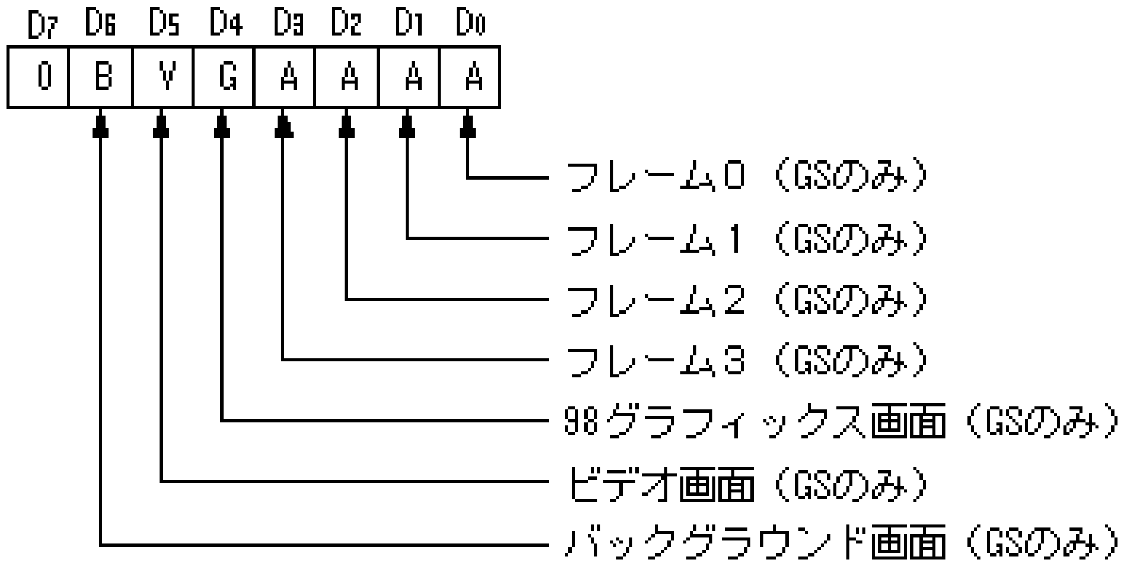
　　00H ： フレーム0(GSのみ)

　　01H ： フレーム1(GSのみ)

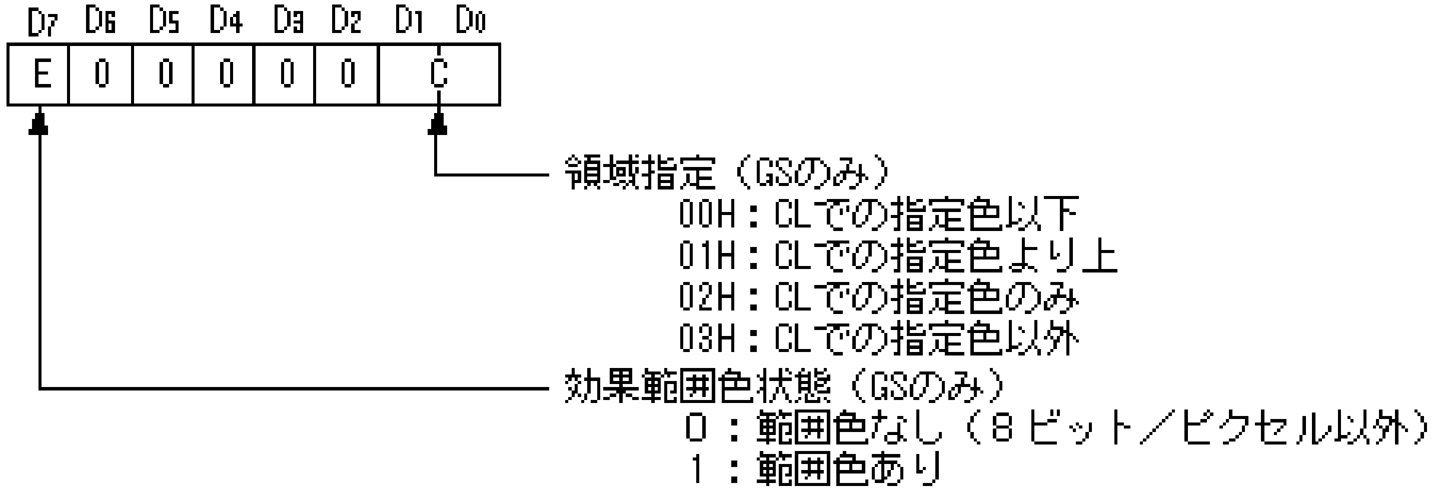
　　02H ： フレーム2(GSのみ)

　　03H ： フレーム3(GSのみ)

BL = 次優先画面



CH ＝ 効果範囲色モード



CL ＝ 効果範囲色(GSのみ)

機　能

現在の特殊効果の設定状態を取得する.

$FADECTRL	AH=17H	

フェードイン/アウト

入　力

AH = 17H

AL = フェードモード

D7	D6	D5	D4	D3	D2	D1	D0
S	G	R	B	E		F	L

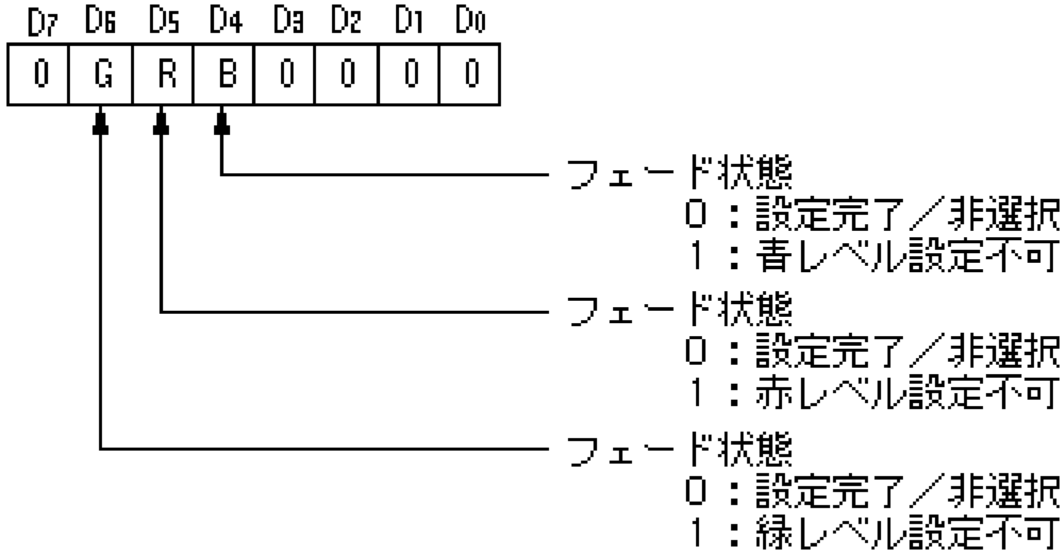
- L (D0): 表示初期値（フェードイン）
  表示最終値（フェードアウト）
  - 0：最低レベル（０）黒
  - 1：最高レベル（２５６）白
- F (D1): フェード方向
  - 0：フェードイン
  - 1：フェードアウト
- E (D3-D2): 実行モード
  - 00：フェード禁止
  - 01：フェード開始
  - 10：フェード中止
  - 11：フェード再開
- B (D4): 実行対象
  - 0：青を対象としない
  - 1：青レベルを対象とする
- R (D5):
  - 0：赤を対象としない
  - 1：赤レベルを対象とする
- G (D6):
  - 0：緑を対象としない
  - 1：緑レベルを対象とする
- S (D7): フェード画面
  - 0：最優先画面
  - 1：第２優先画面

CX ＝ フェードスピード1μ秒単位で指定
1μsec～26.666msecの範囲で指定
（～32.82msecまではエラーにならないが，設定禁止）

出　力

Carry ＝ 実行ステータス
0 ： 正常設定
1 ： エラー

AL ＝ フェードステータス（Carry=0，入力：AL(3)=1）
中止/再開のみのステータス



AH ＝ エラーステータス（Carry=1）
1 ： 特殊効果状態でない
2 ： フェード機能禁止のため，再開不能
3 ： フェードスピード範囲外

機　能

実行対象で指定された色に対するフェード処理が行われる.

フェードイン/アウトは特殊効果の一部であり，「特殊効果の設定[表示優先順位依存](AH=14H)」，「特殊効果の設定[優先順位指定](AH=15H)」によって特殊効果が設定されている必要がある.

ハードウェアは指定されたフェードレベルと画面データの加減算を行い，割り込みを用いて一定間隔でフェードレベルを操作することによりフェードイン/アウトを実現する.

$ASKFADE	AH=18H	

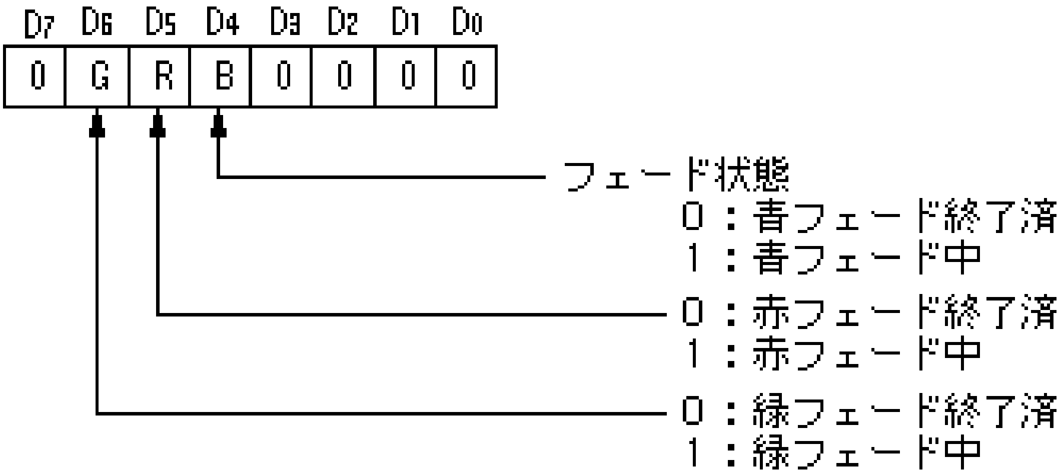
フェードレベルの取得

入　力

AH = 18H
AL = 対象画面指定
　　00H ： 最優先画面
　　80H ： 第2優先画面

出　力

Carry = 獲得ステータス
　　0 ： 正常獲得
　　1 ： 特殊効果状態でない
　　　　フェードなし
AL = フェードステータス(Carry=0)



BL = 青フェードレベル
CH = 赤フェードレベル
CL = 緑フェードレベル
　黒からのフェードイン時, 0 → 255へ変化する
　白からのフェードイン時, 255 → 0へ変化する
　黒へのフェードアウト時, 255 → 0へ変化する
　白へのフェードアウト時, 0 → 255へ変化する

機　能

現在のフェードイン/アウトの状態を取得する.

フェードイン/アウトは割り込みで処理されるため, フェードレベルはこのファンクション実行後も刻々と変化する.

したがって, 正確なフェードレベルを求めるには, 「フェードイン/アウト(AH=17H)」にて一旦フェード処理を中断する必要がある.

$START	AH=19H	
グラフィックスドライバの使用開始宣言		

入　力

AH = 19H

出　力

なし

機　能

グラフィックスドライバの使用開始を宣言する.

本ファンクションは，マスク機能/透明色/フェード/特殊効果/ビデオ画面の初期化と，テキストと98グラフィックス画面の1ドットズレを保証することはできない.

注　意

本ファンクションを使用しないと他の拡張グラフィックス機能を使用できない.

$END	AH=1AH	
グラフィックスドライバの使用終了宣言		

入　力

AH = 1AH

出　力

Carry = 同期ステータス
　0 ： 正常終了
　1 ： エラー
AH = エラーステータス
　1 ： パラメータ異常
　2 ： 同期タイミングオーバーラン

機　能

グラフィックスドライバの使用終了を宣言する.

本ファンクションは，「水平周波数24KHz，テキスト25行，グラフィックス200ライン」に設定し，拡張画面の表示禁止と，テキストと98グラフィックス画面の1ドットズレを行う.

注　意

本ファンクションを使用しないと従来アプリケーションが動作しなくなることがある.

$GETINFO	AH=1BH	GS

グラフィックスドライバの機能の取得

入　力

AH = 1BH
AL = 0 ： データテーブル長の取得
　　 1 ： 概要取得
　　 2 ： 詳細取得
CX = データテーブル長(AL = 1または2の場合のみ)
ES:BX = データテーブル転送アドレス

出　力

Carry = 同期ステータス
　　　0 ： 正常終了
　　　1 ： エラー
CX = データテーブル格納サイズ(AL = 0のとき)
ES:BX = データテーブル格納アドレス(AL = 1，2のとき)

機　能

現在使用しているグラフィックスドライバの機能を取得する.

ALに0が指定された場合，データテーブルのサイズをバイトサイズでCXに返す.

ALに1が指定された場合，ES:BXで指定されたアドレスからのメモリ領域に機能データ(概要)が転送される.

ALに2が指定された場合，ES:BXで指定されたアドレスからのメモリ領域に機能データ(詳細)が転送される.

注　意

ALに1を指定した概要取得では，拡張グラフィックスドライバの最大機能を取得する．

取得できる機能はあくまでも最大の機能であり，画面の状況により機能が低下している場合もある．

取得データ例

バージョン番号	0101H
最大サポート画面モード	0108H
予約	0000H
拡張機能	0003H
予約	0000H
合成表示可能画面数	0001H
最大パレット数	0001H
最大定義画面数	0002H

ALに2を指定した現状取得では，拡張グラフィックスドライバの現状を機能が取得する．

取得できる機能は，現在の画面モードに従ったものであり，画面モードが変わると情報は変化する．したがって，画面モードを変更した場合，アプリケーションは本ファンクションを再度実行し，正しい画面情報を参照しなければならない．

データテーブル

転送されるデータテーブルは，以下のような階層構造をもつ．ただし，ALに1を指定して概要を取得した場合は，第1階層の「バージョン番号(オフセット+00H)」から「最大定義画面数(オフセット+0EH)」までのデータしか取得されない．

▼*戻り値のデータ構造=ES:BX〔戻り値〕(INT 0D8H/1BH)*

●第1階層

▼*第1階層のテーブル〔戻り値〕(INT 0D8H/1BH)*

・バージョン番号
　拡張グラフィックスドライバのバージョン番号が格納される．

・最大サポート画面モード
　サポートしている画面モードの各ビットが1となる．

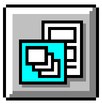
▼*最大サポート画面モード〔戻り値〕(INT 0D8H/1BH)*

PC-9821では　“0000000100001000”となる．

・拡張機能

サポートしている拡張機能を各ビットで表す.

▼*拡張機能〔戻り値〕（INT 0D8H/1BH）*



PC-9821では “0000000000000011”となる.

・合成表示可能画面数

拡張グラフィックスドライバがサポートしている画面のうちで合成可能なものの枚数を表す．テキスト/98グラフィックス等，従来のBIOSもしくはLIOでサポートしているような画面は枚数には含まれない．ただし，拡張グラフィックスドライバが従来のLIO等を包含してサポートし，面合成が任意に組み合わせができる場合は画面枚数にカウントされる.

PC-9821では1枚.

・最大パレット数

独立して使用できるパレットの総数を表す.

PC-9821では1.

・最大定義画面数

定義可能な画面の総数を表す.

PC-9821では2枚.

・画面情報テーブル

現在確保されている画面の情報を示す.

最大画面数分の画面情報テーブルが存在する.

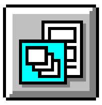
第2階層として展開される.

ALに1を指定して概要を取得した場合は存在しない.

●第2階層(画面情報テーブル)

第2階層は現在確保されている画面の情報が格納されている.

▼*第2階層(画面情報)のテーブル〔戻り値〕(INT 0D8H/1BH)*



・画面情報テーブルサイズ

画面情報テーブル(第2階層)のバイトサイズ.

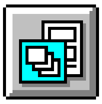
・フレーム番号

画面情報テーブル(第2階層)が示すフレーム(画面)の番号を表す(0から“最大定義画面数-1”まで).

・画面モード

画面情報テーブル(第2階層)が示すフレームの画面モードを表す.

▼*画面モード〔戻り値〕(INT 0D8H/1BH)*

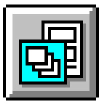


内容は「最大サポート画面モード」と同様.値が0の場合は,現在使用されていないフレームである(そのフレーム番号,色数,参照パレット番号,フレームの幅,フレームの高さ,ウィンドウの数,ウィンドウ情報は意味をもたない).

PC-9821では“0000000100000000”となる.

・拡張機能
画面情報テーブル(第2階層)が示すフレームの拡張機能を各ビットで表す.

▼拡張機能〔戻り値〕（INT 0D8H/1BH）



PC-9821では“00000000000000?1”となる.

・色数
画面情報テーブルが示すフレームの色数を表す.
PC-9821では“256”となる.

・参照パレット番号
画面情報テーブル(第2階層)が示すフレームが参照しているパレット番号を表す.

・フレームの幅
画面情報テーブル(第2階層)が示すフレームの横幅をピクセル数で表す.
PC-9821では640ピクセルである.

・フレームの高さ
画面情報テーブル(第2階層)が示すフレームの高さをピクセル数で表す.
PC-9821では400/480である.

・1ラインのバイト数
画面情報テーブル(第2階層)が示すフレームの横幅をバイト数で表す.
PC-9821では640バイトである.

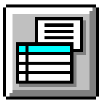
・ウィンドウの数
画面情報テーブル(第2階層)が示すフレームの拡張グラフィックスVRAMをアクセスするウィンドウの使用可能数を示す.

・ウィンドウ情報
画面情報テーブル(第2階層)が示すフレームの拡張グラフィックスVRAMをアクセスするウィンドウの情報を示す.
ウィンドウの数のウィンドウ情報が存在する.
第3階層として展開される.

●第3階層(ウィンドウ情報)

第3階層は，画面情報テーブル(第2階層)が示すフレームの拡張グラフィックスVRAMをアクセスするウィンドウの情報が格納されている.

▼*第3階層(ウィンドウ情報)のテーブル〔戻り値〕(INT 0D8H/1BH)*



・ウィンドウアクセス方式

画面情報テーブル(第2階層)が示すフレームの拡張グラフィックスVRAMをアクセスするウィンドウのアクセス方式を示す.

0 ： 不定(ウィンドウ領域が正常に初期化されていない)
「グラフィックスドライバの使用開始宣言(AH = 19H)」および「グラフィックスドライバ使用終了宣言(AH = 1AH)」を実行しなおす必要がある.

1 ： ウィンドウ I/O選択方式
ウィンドウ領域をI/Oを用いて選択する.

2 ： ウィンドウメモリマップトI/O選択方式(PC-9821)
ウィンドウ領域をメモリマップトI/Oを用いて選択する.

・ウィンドウのアドレス
画面情報テーブル(第2階層)が示すフレームの拡張グラフィックスVRAMをアクセスするウィンドウの32ビットアドレスを示す.
PC-9821では以下の2つのうちいずれかをとる.

ウィンドウ#0	000A : 8000H
ウィンドウ#1	000B : 0000H

・ウィンドウのバイトサイズ
画面情報テーブル(第2階層)が示すフレームの拡張グラフィックスVRAMをアクセスするウィンドウ1つのサイズをバイトで示す.
PC-9821では32768バイトである.

・ウィンドウのアクセス単位
画面情報テーブル(第2階層)が示すフレームの拡張グラフィックスVRAMをアクセスするウィンドウの設定単位をバイトで示す.
PC-9821では32768バイトである.

・ウィンドウ設定アドレス
画面情報テーブル(第2階層)が示すフレームのAVGRAMをアクセスするウィンドウを設定するI/Oアドレスもしくはメモリマップト I/Oアドレスを示す.
PC-9821では以下の2つのうちいずれかをとる.

ウィンドウ#0	000EH : 0004H
ウィンドウ#1	000EH : 0006H

第3章

ビデオ系ファンクション

$DIGITIZE	AH=20H	GS
デジタイズの開始/停止		

入　力

AH ＝ 20H
AL ＝ デジタイズ制御
　　0 ： デジタイズ停止
　　1 ： デジタイズ開始

出　力

Carry ＝ 実行ステータス
　　0 ： 正常動作
　　1 ： エラー
AH ＝ エラーステータス
　　1 ： パラメータエラー
　　2 ： フレーム定義不良
　　3 ： ビデオボード未実装する

機　能

ビデオ信号のデジタイズを開始する．

デジタイズが必ずフレーム0に対し実行される．

ビデオボードが実装されていない場合はエラー(Carry=1，AL=3)を返す．デジタイズ対象画面(フレーム0)が8ビット/ピクセルの場合，デジタイズ機能は動作しないため，エラー(Carry=1，AL=2)を返す．

$ASKDIGI	AH=21H	GS

デジタイズ状況の取得

入　力

AH = 21H

出　力

Carry = 実行ステータス
　　0 ： 正常動作
　　1 ： ビデオボード未実装
AH = エラーステータス
　　3 ： ビデオボード未実装
AL = デジタイズ制御（Carry=0の場合）
　　0 ： 非デジタイズ
　　1 ： デジタイズ中

機　能

現在のデジタイズ状況を取得する.

$SHARP	AH=22H	

画質補正の設定

入　力

AH = 22H
AL = 輪郭強調設定
　　0 ： 強調なし
　　1 ： 強調あり

出　力

Carry = 実行ステータス
　　0 ： 正常動作
　　1 ： エラー
AH = エラーステータス
　　3 ： ビデオボード未実装

機　能

ビデオ画面の輪郭強調機能をオン/オフする.
輪郭画質を補正することができる.
ビデオボードが実装されていない場合はエラー(Carry=1)を返す.

$ASKSHARP	AH=23H	
画質補正状態の取得		

入　力

AH = 23H

出　力

Carry = 実行ステータス
　　0 ： 正常動作
　　1 ： エラー
AH = エラーステータス
　　3 ： ビデオボード未実装
AL = 輪郭強調状態（Carry=0の場合）
　　0 ： 強調なし
　　1 ： 強調あり

機　能

現在のビデオ画面の輪郭強調状態を取得する.

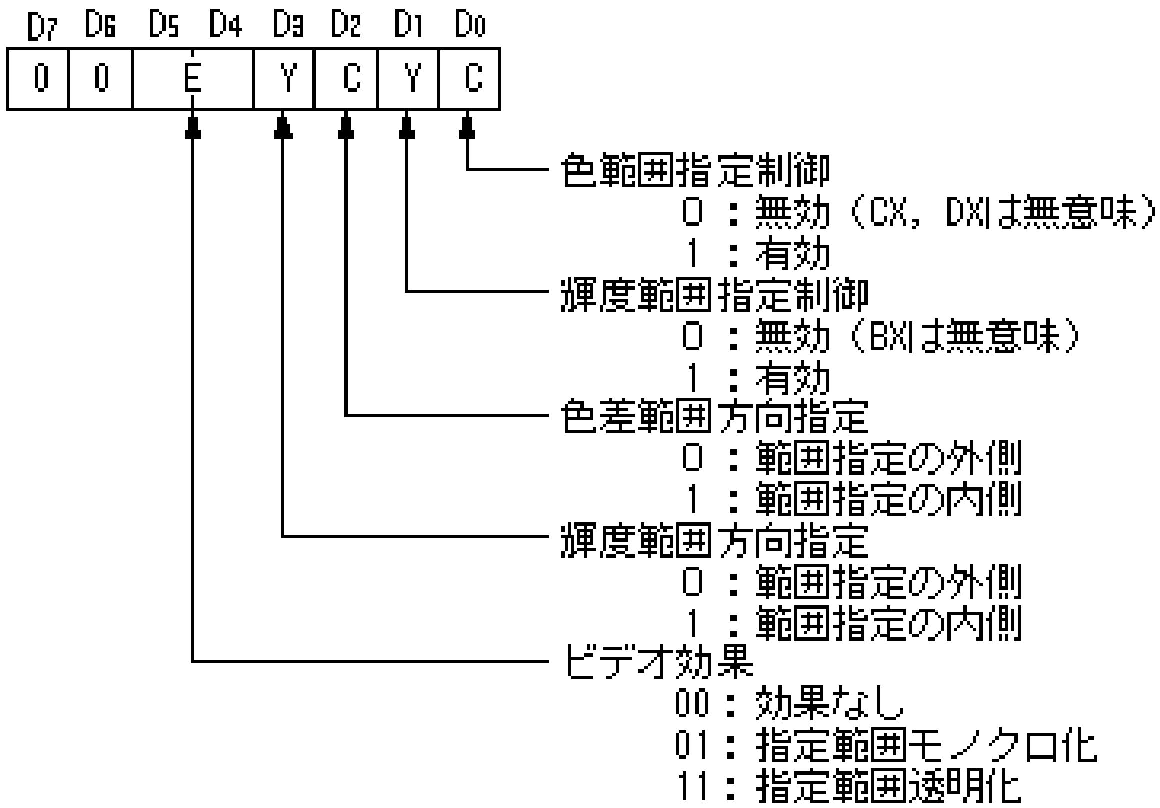
$V_EFECT	AH=24H	

ビデオ特殊効果の設定

入　力

AH ＝ 24H

AL ＝ 効果モード



BH ＝ 輝度信号(Y)上限レベル（0〜255）

BL ＝ 輝度信号(Y)下限レベル（0〜255）

CH ＝ 色差信号(R−Y)上限レベル（−128〜127）

CL ＝ 色差信号(R−Y)下限レベル（−128〜127）

DH ＝ 色差信号(B−Y)上限レベル（−128〜127）

DL ＝ 色差信号(B−Y)下限レベル（−128〜127）

出　力

Carry = 実行ステータス
　0 ： 正常設定
　1 ： エラー
AH = エラーステータス
　3 ： ビデオボード未実装

機　能

指定された範囲に該当するレベルの信号に対し効果を加える.
効果には次の2つがある.

・色差信号を0レベルにする. → モノクロ化
・色差信号, 輝度信号共に0レベルにする. → 透明化

ビデオボードが実装されていない場合はエラー(Carry=1)を返す.

$ASK_ VEF	AH=25H	

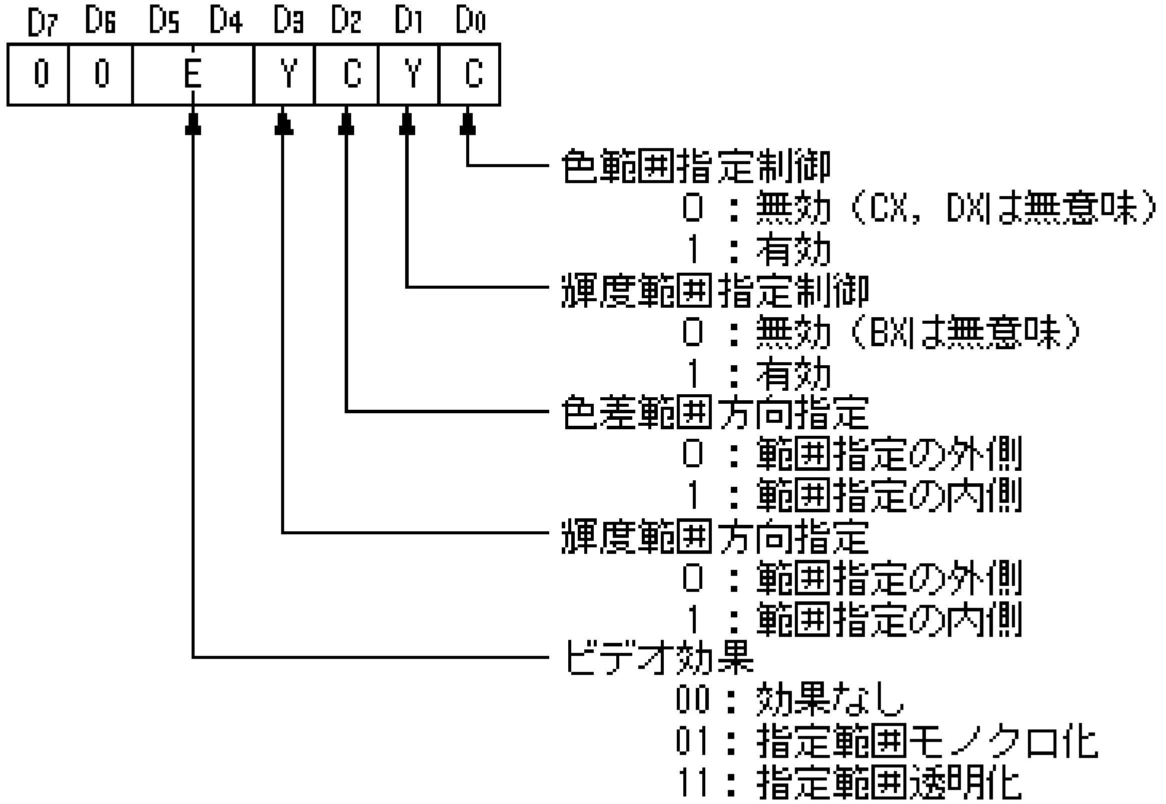
ビデオ特殊効果状態の取得

入　力

AH = 25H

出　力

Carry = 実行ステータス
　0 ： 正常動作
　1 ： エラー
AH = エラーステータス
　3 ： ビデオボード未実装
AL = 効果モード



BH = 輝度信号(Y)上限レベル (0～255)
BL = 輝度信号(Y)下限レベル (0～255)
CH = 色差信号(R-Y)上限レベル (-128～127)
CL = 色差信号(R-Y)下限レベル (-128～127)
DH = 色差信号(B-Y)上限レベル (-128～127)
DL = 色差信号(B-Y)下限レベル (-128～127)

機　能

現在のビデオ特殊効果状態を取得する.

$HUE	AH=26H	
彩度の調整		

入　力

AH = 26H
AL = 彩度補正レベル(-128～127)

出　力

Carry = 実行ステータス
　0 ： 正常動作
　1 ： エラー
AH = エラーステータス
　3 ： ビデオボード未実装

機　能

ビデオ画面の彩度(色相)補正レベルをビデオボード上のボリュームに与える.

ビデオボードが実装されていない場合はエラー(Carry=1)を返す.

$COLOR	AH=27H	
濃度の調整		

入　力

AH = 27H
AL = 濃度補正レベル(-128～127)

出　力

Carry = 実行ステータス
　　　0 ： 正常動作
　　　1 ： エラー
AH = エラーステータス
　　3 ： ビデオボード未実装

機　能

ビデオ画面の濃度(濃さ)補正レベルをビデオボード上のボリュームに与える.

ビデオボードが実装されていない場合はエラー(Carry=1)を返す.

$BRIGHT	AH=28H	
明度の調整		

入　力

AH = 28H
AL = 明度補正レベル(-128～127)

出　力

Carry = 実行ステータス
　　0 ： 正常動作
　　1 ： エラー
AH = エラーステータス
　　3 ： ビデオボード未実装

機　能

ビデオ画面の明度(明るさ)補正レベルをビデオボード上のボリュームに与える.

ビデオボードが実装されていない場合はエラー(Carry = 1)を返す.

$CONTRAST	AH=29H	

照度の調整

入　力

AH = 29H
AL = 照度補正レベル(-128～127)

出　力

Carry = 実行ステータス
　0 ： 正常動作
　1 ： エラー
AH = エラーステータス
　3 ： ビデオボード未実装

機　能

ビデオ画面の照度(コントラスト)補正レベルをビデオボード上のボリュームに与える．ビデオボードが実装されていない場合はエラー(Carry=1)を返す．

$ASKLEVEL	AH=2AH	

彩度/濃度/明度/照度の取得

入　力

AH = 2AH

出　力

Carry = 実行ステータス
0 ： 正常動作
1 ： エラー
AH = エラーステータス
3 ： ビデオボード未実装
BH = 彩度(色相)補正レベル(−128～127)
BL = 濃度(濃さ)補正レベル(−128～127)
CH = 明度(明るさ)補正レベル(−128～127)
CL = 照度(コントラスト)補正レベル(−128～127)

機　能

現在のビデオ画面の彩度，濃度，明度，照度を取得する．

$V_WINDOW	AH=2CH	
表示枠と位置の設定/取得		

入　力

AH = 2CH
AL = 動作モード(設定/取得)
　　0 ： 設定
　　1 ： 取得
ES:BX = 表示パラメータテーブルアドレス(設定時)

出　力

Carry = 実行ステータス
　　0 ： 正常動作
　　1 ： エラー
AH = エラーステータス
　　1 ： パラメータエラー
　　3 ： ビデオボード未実装
ES:BX = 表示パラメータテーブルアドレス(取得時)

パラメータテーブル

ES：BX

TYPE κ

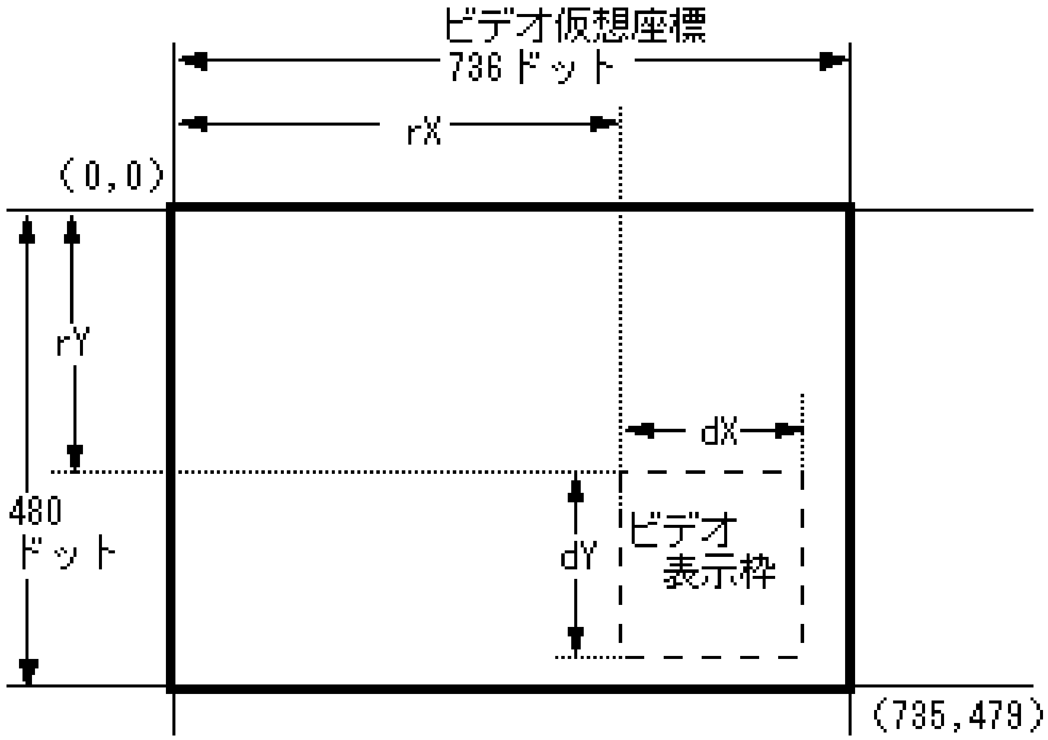
sX	sY	dX	dY	rX	rY

sX ： 表示水平座標(ディスプレイ座標で指定する)
sY ： 表示垂直座標
dX ： 表示水平サイズ(ビデオ仮想座標で指定する)
dY ： 表示垂直サイズ
rX ： 表示開始水平座標(ビデオ仮想座標で指定する)
rY ： 表示開始水平座標

機　能

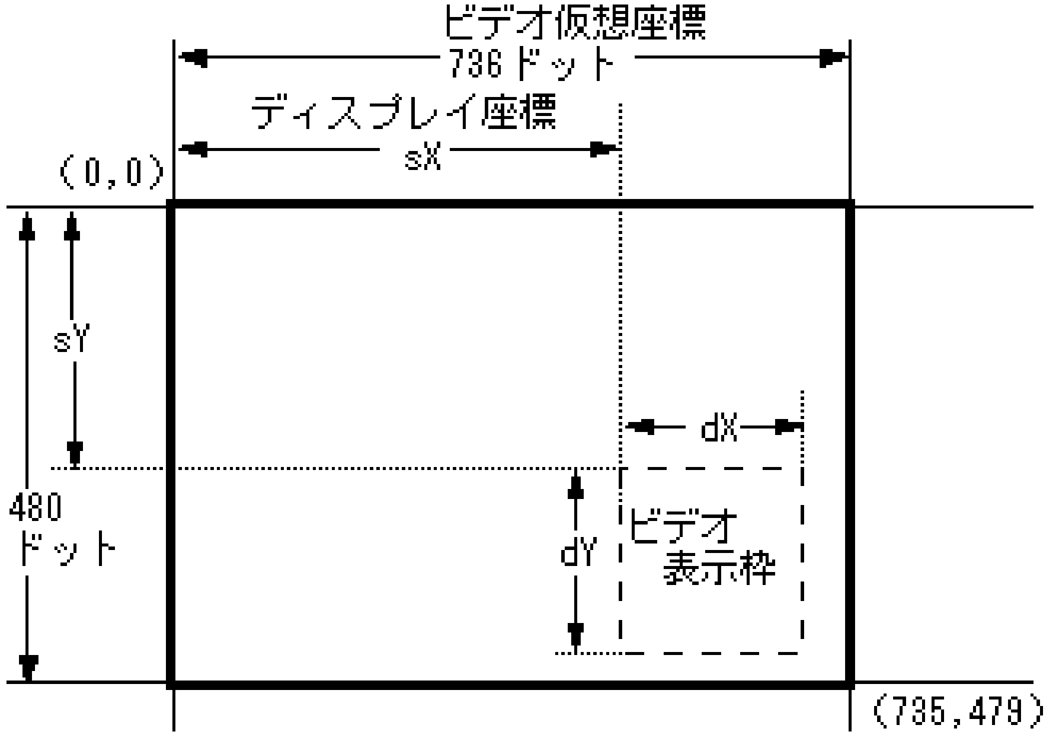
ビデオ画面の表示枠と位置を設定もしくは取得する.

以下に表示枠の概念図を示す. ビデオ仮想座標は, 水平周波数に関係なく736×480の解像度である. 下図の場合網掛けの矩形部分のみ画面に表示されるようになる.

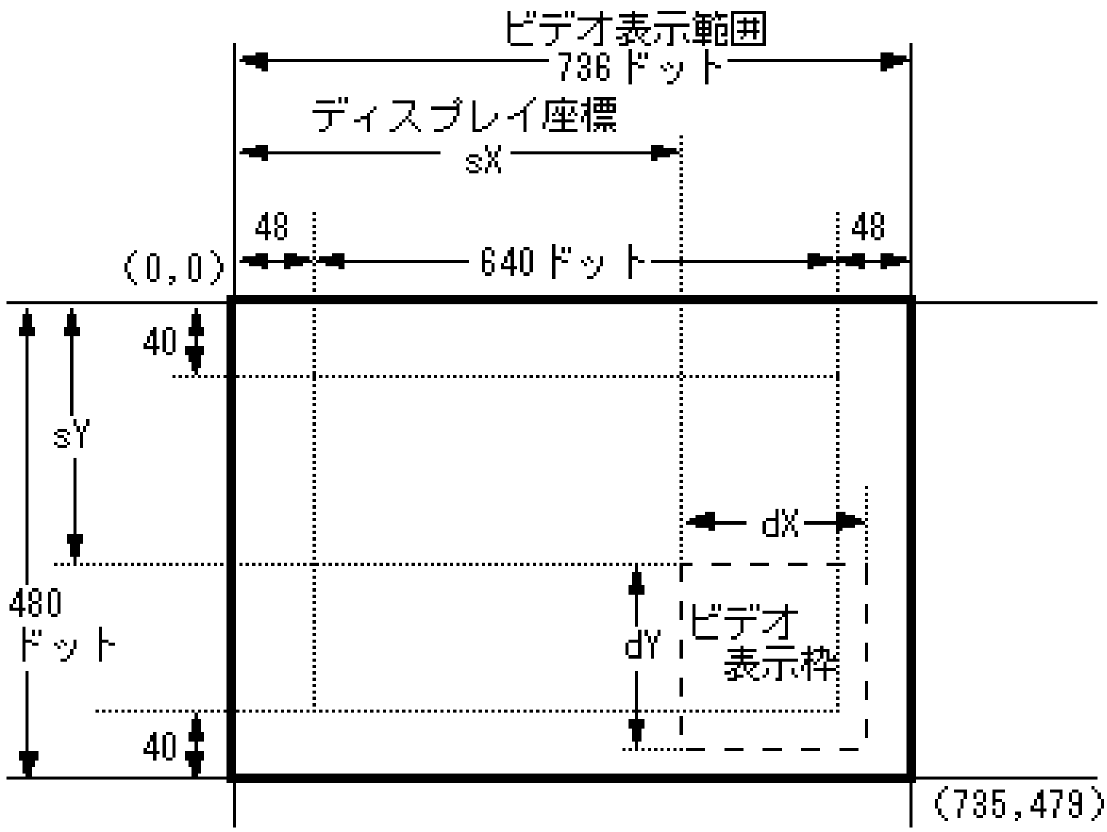


以下に各周波数時の表示位置の概念図を示す. 表示座標は, 水平周波数により解像度が変化する.

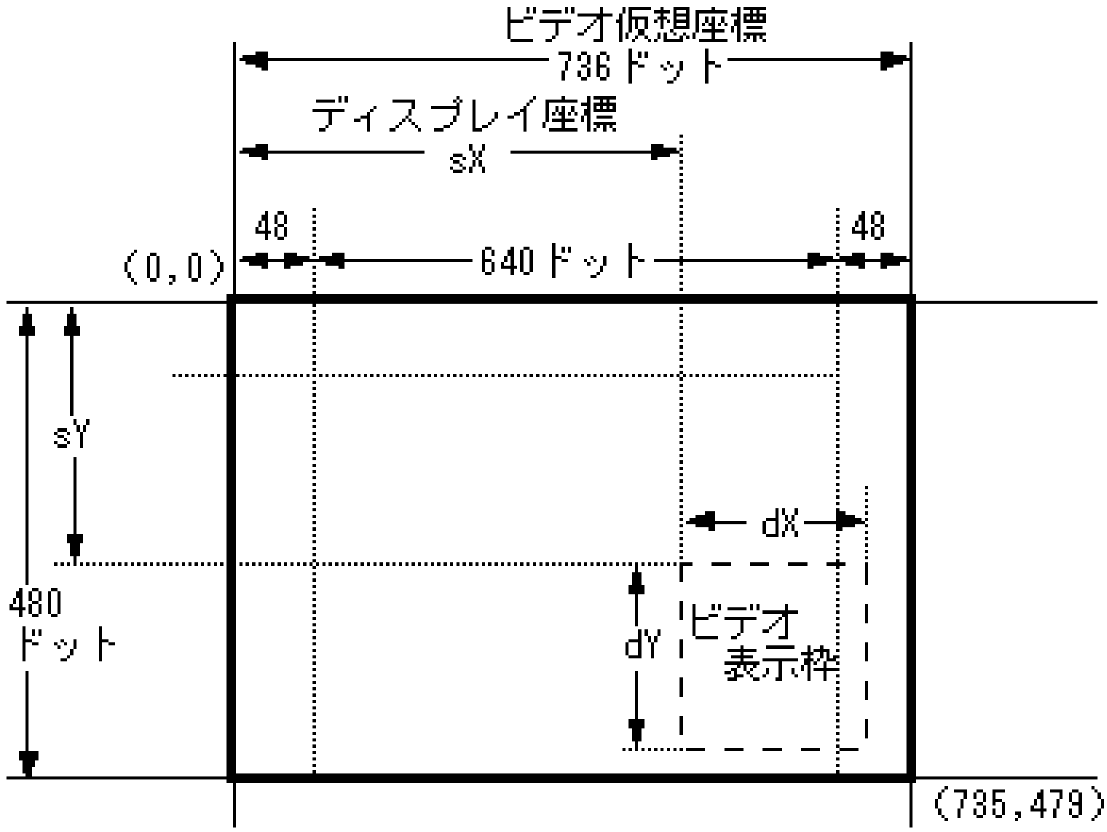
・15KHz ： ディスプレイ表示範囲(GSのみ)



・24KHz ： ディスプレイ表示範囲



・31KHz ： ディスプレイ表示範囲



V_SIZE	AH=2DH	
縮小サイズの設定/取得		

入　力

AH = 2DH
AL = 動作モード(設定/取得)
　　0 ： 設定
　　1 ： 取得
BX = 水平サイズ(1～736)(設定時のみ)
CX = 垂直サイズ(1～480)(設定時のみ)

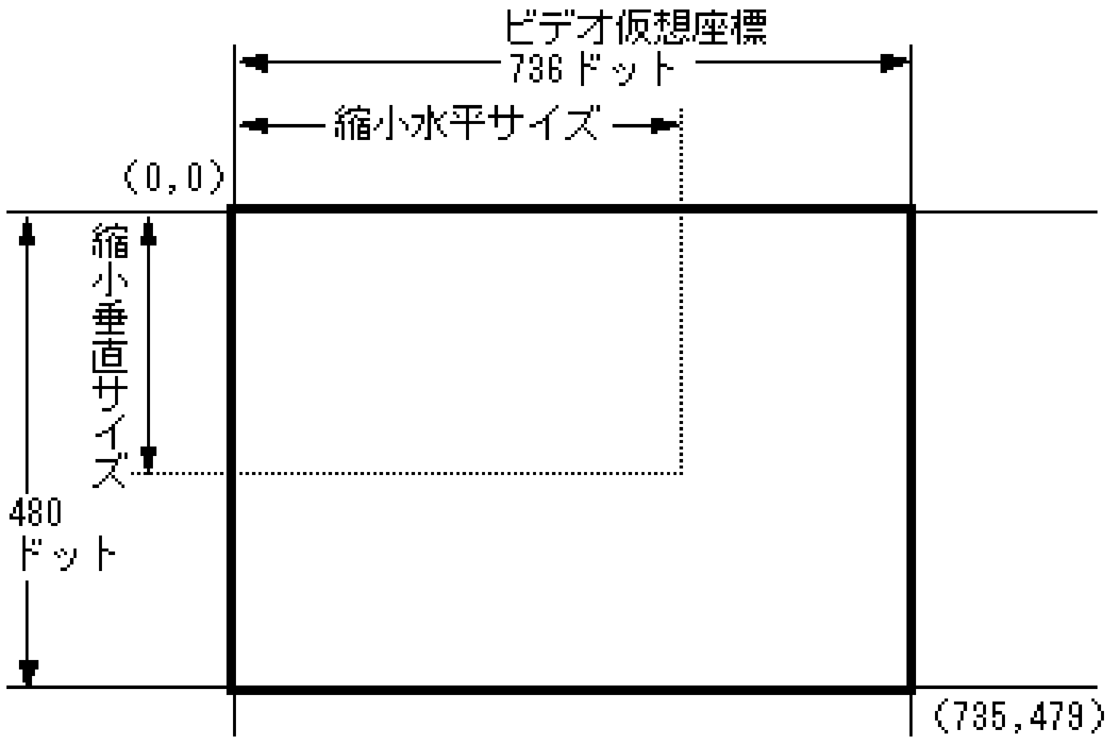
出　力

Carry = 実行ステータス
　　0 ： 正常動作
　　1 ： エラー
AH = エラーステータス
　　1 ： パラメータエラー(設定時のみ)
　　3 ： ビデオボード未実装
BX = 水平サイズ(取得時のみ)
CX = 垂直サイズ(取得時のみ)

機　能

ビデオ画面の縮小サイズを設定もしくは取得する．

以下に縮小時の概念図を示す．ビデオ仮想座標は，水平周波数に関係なく736×480の解像度である．ビデオイメージは，縮小サイズに関係なく常に左上の原点からビデオボードRAMに書き込まれる．



$V_PAT	AH=2FH	
ビデオボードRAMの読み出し/書き込み		

入　力

AH = 2FH
AL = 動作モード
　　0 ： 読み出し
　　1 ： 書き込み
ES:BX = 転送パラメータテーブルアドレス

出　力

Carry = 実行ステータス
　　0 ： 正常動作
　　1 ： エラー
AH = エラーステータス
　　1 ： パラメータエラー
　　3 ： ビデオボード未実装

パラメータテーブル

ES：BX
↓

sX	sY	dX	dY	Doff	Dseg	RGB

sX ： 転送水平座標(ビデオ仮想座標で指定する)
sY ： 転送垂直座標
dX ： 転送水平サイズ(ビデオ仮想座標で指定する)
dY ： 転送垂直サイズ
Doff ： データ格納オフセットアドレス(本体メモリアドレス)
Dseg ： データ格納セグメントアドレス
RGB ： 0 ： 青データのみ読み出し/書き込み
　　　 1 ： 赤データのみ読み出し/書き込み
　　　 2 ： 緑データのみ読み出し/書き込み
　　　 3 ： 青赤緑すべての読み出し/書き込み
　　　 上記以外の設定は禁止

機　能

ビデオボードRAMの読み出し，書き込みを行う．

64Kバイトを超えるデータ転送が可能である．

1ピクセルは，RGB各1バイト(1677万色で構成されている)．

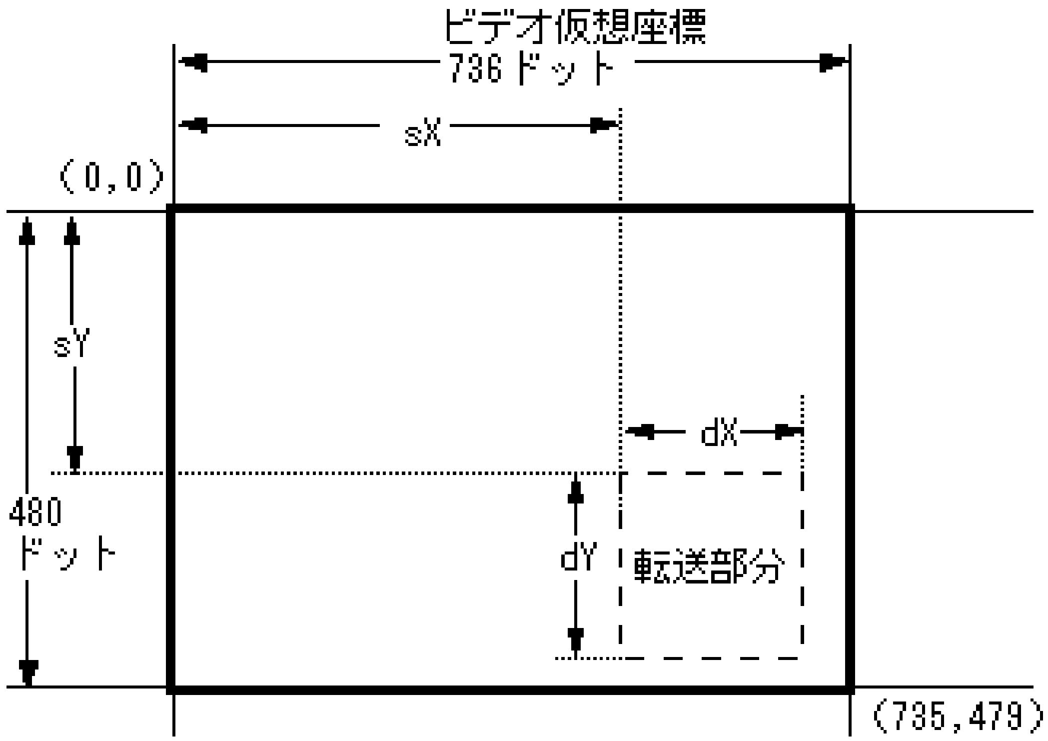
転送パラメータテーブルのRGBパラメータで0を指定したときのデータ配列を示す．

1Dot	1Dot	1Dot	1Dot	1Dot	1Dot	1Dot	1Dot
青	青	青	青	青	青	青	青

転送パラメータテーブルのRGBパラメータで3を指定したときのデータ配列を示す．

1Dot	1Dot	1Dot	1Dot	1Dot	1Dot		
青	赤	緑	青	赤	緑	青	赤

以下に座標系の概念図を示す．ビデオ仮想座標は，水平周波数に関係なく736×480の解像度である．



注　意

ビデオボードRAM読み出し時には，必ず「スチル/フレームバッファの設定/取得(AH = 91H)」でスチルモードに設定しなければならない．

また，書き込み時には，必ずフレームバッファモードに設定しなければならない．

転送水平/垂直位置は，表示位置や水平周波数に関係なくビデオ仮想座標で指定すること．また，ビデオボードRAMの書き込み時は，“転送水平位置(sx)/サイズ(sy)”は，60の倍数 でなければならない．

・ビデオボードRAMに書き込み時に指定できる値

転送水平位置(sx)

0，60，120，180，240，300，360，

420，480，540，600，660，720

転送水平サイズ(sy)

60，120，180，240，300，360，

420，480，540，600，660，720，780

また，転送水平位置(sx)と転送水平サイズ(sy)を足した値が780を超えてはいけない．

$FIELD	AH=90H	

フィールドモードの設定/取得

入　力

AH = 90H
AL = 動作モード
　0:設定
　1:取得
BH = フィールドモード(設定時のみ)
　0:1フィールドモード
　1:2フィールドモード

出　力

Carry = 設定ステータス
　0:正常終了
　1:ビデオボード未実装
AH = エラーステータス
　1:パラメータエラー(設定時のみ)
　3:ビデオボード未実装
BH = フィールドモード(取得時のみ)
　0:1フィールドモード
　1:2フィールドモード

機　能

フィールドモードの設定，取得を行う．

フィールドモードには，1フィールドモードと2フィールドモードの2種類がある．

1フィールドモードは，偶数ラインと奇数ラインに同じデータが表示される．そのため画面が，ちらつきにくい．

2フィールドモードは，偶数ラインと奇数ラインで違うデータ（実際はビデオの解像度が低いためほとんど同じ）が表示される．1フィールドモードに比べるとちらつきやすい．しかし解像度は高いので，画像によっては1フィールドモードよりきれいに表示される．

$STEAL	AH=91H	
ビデオボード動作モードの設定/取得		

ビデオボード動作モードの設定/取得

入　力

AL = 動作モード
　　0:設定
　　1:取得
BH = スチルモード(設定時のみ)
　　00:動画モード
　　01:スチルモード
　　10:フレームバッファ
　　11:設定禁止

出　力

Carry = 設定ステータス
　　0:正常終了
　　1:エラー
AH = エラーステータス
　　1:パラメータエラー(設定時のみ)
　　3:ビデオボード未実装
BH = モード(取得時のみ)
　　00:動画モード
　　01:スチルモード
　　02:フレームバッファモード
　　03:設定禁止

機　能

ビデオボードの動画モードを設定取得する.
ビデオボードには以下の3つの動作モードがある.

●動画モード
リアルタイムに入力されたビデオ信号を表示している状態. ビデオボードのフレームバッファには, リアルタイムにビデオ信号がY色差形式でデジタイズされる.

●スチルモード
ビデオボードのフレームバッファの画像データをY色差形式として表示する. ビデオ信号のデジタイズは行われないため, 動画モードからスチルモードにするとスチルモードになったときのビデオ画面が表示される.

●フレームバッファモード
ビデオボードのフレームバッファの画像データをRGB形式として表示する. ビデオ信号のデジタイズは行われないため, 動画モードからフレームバッファモードにするとフレームバッファモードになったときのビデオ画像データをRGB形式として表示する（ビデオ画像データはY色差形式のため画面が乱れる）.

注　意

「ビデオボードRAMの読み出し/書き込み(AH = 2FH)」を使うときには, スチルもしくはフレームバッファモードにしなければならない.

$FILTER	AH=92H	GS

画面ズレの補正値の設定/取得

入　力

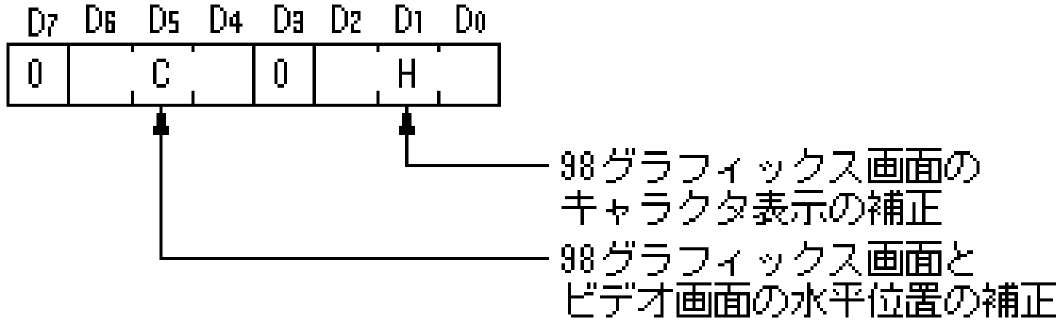
AH = 92H

AL = 動作モード

　　0:設定

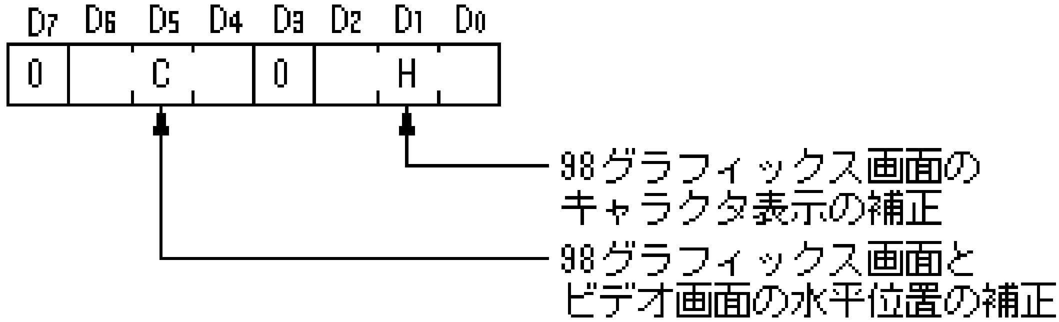
　　1:取得

BH = 補正値(設定時のみ)



出　力

Carry = 設定ステータス
　　　　0:正常終了
　　　　1:エラー
AH = エラーステータス(設定時のみ)
　　1:パラメータエラー
　　3:ビデオボード未実装
BH = 補正値(取得時のみ)



機　能

画面のズレ補正値を設定，取得する．

$G_THRU	AH=93H	GS

98グラフィックス画面の透明レベルの設定/取得

入　力

AH = 93H

AL = 動作モード/透明設定

D7	D6	D5	D4	D3	D2	D1	D0
0	0	C	0	G	R	B	M

- M（D0）：動作モード
  - 0：設定
  - 1：取得
- B（D1）：透明色の範囲（設定時のみ）
  - 0：青の上限下限の外側を透明色
  - 1：青の上限下限の内側を透明色
- R（D2）
  - 0：赤の上限下限の外側を透明色
  - 1：赤の上限下限の内側を透明色
- G（D3）
  - 0：緑の上限下限の外側を透明色
  - 1：緑の上限下限の内側を透明色
- C（D5）：透明指定（設定時のみ）
  - 0：透明の指定をする
  - 1：透明色の指定をする

透明モードで透明レベルの指定をする場合，以下のパラメータで透明レベルを指定する.

BH = 青上限レベル
BL = 青下限レベル
CH = 赤上限レベル
CL = 赤下限レベル
DH = 緑上限レベル
DL = 緑下限レベル

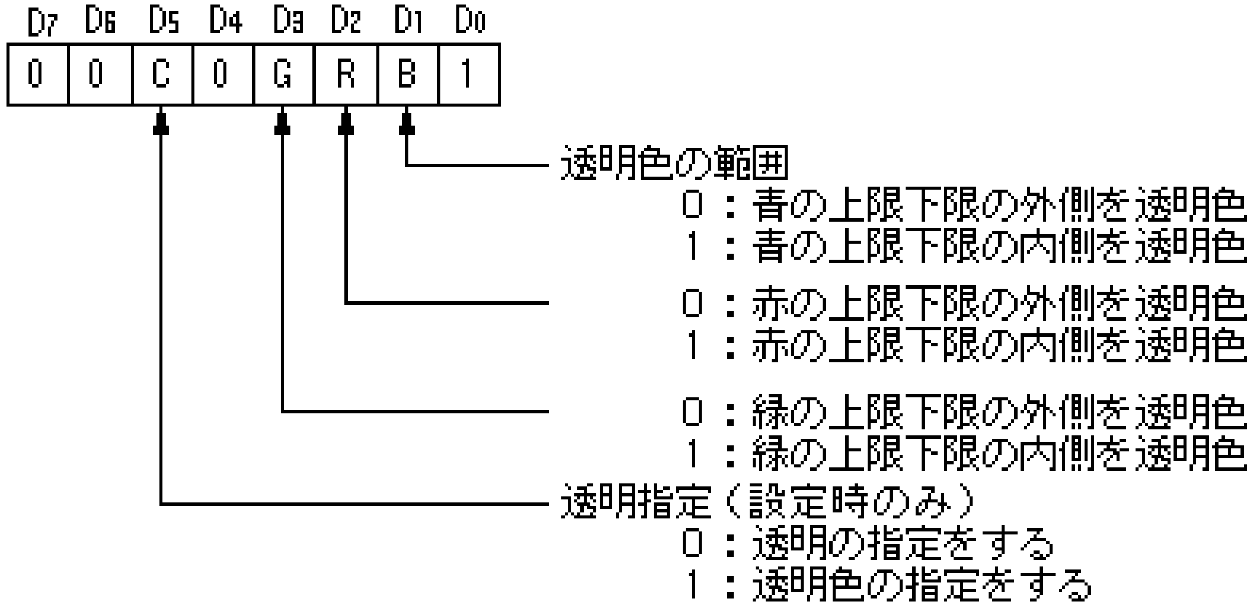
（以上6項目）…1677万色で指定する(注意参照)

出　力

Carry = 設定ステータス
　0:正常終了
　1:エラー

AH = エラーステータス
　1:パラメータエラー(設定時のみ)
　3:ビデオボード未実装

AL = 透明指定(取得時のみ)



BH = 青上限レベル
BL = 青下限レベル
CH = 赤上限レベル
CL = 赤下限レベル
DH = 緑上限レベル
DL = 緑下限レベル

（BH～DL）…1677万色で返される

機　能

98グラフィックス画面(16色モード)の透明レベルを設定，取得する．

透明モードの指定で，透明レベルの指定をしない(0)を選択した場合は，指定した透明レベルと透明色の範囲により以下のように透明色が決まる．

透明色はRGBそれぞれ個々に次のように透明色の範囲が決定される．

透明範囲の指定が内側の場合はその色の透明レベルの下限から上限の間が透明色になる．透明範囲の指定が外側の場合は，その色の透明レベルの下限から0までと上限からFFまでの間が透明になる．

透明モードの指定で，透明レベルの指定をしない(0)を選択した場合は，黒(0)が透明色となる．

注　意

本ファンクションは，GSでは使用できない．

グラフィックス画面が標準モードの場合，パレットが4096色のサポートとなるため，1677万色に換算したパレットレベルを用いなければならない．

	青	赤	緑
カラー番号1のパレット	1100	0010	1001
1677万色でのパレット	11001100	00100010	10011001

また，ビデオボードはアナログ信号をデジタル化した後，ビデオ画面との1画面合成を行っている．このため，本体のA/Dコンバータのばらつきによりうまく合成できない場合がある．したがって，色の上限/下限指定はある程度設定に幅を持たせておくことが望ましい．

$V_CLS	AH=94H	
フレームバッファの高速なクリア		

入　力

AH = 94H

出　力

Carry = 設定ステータス
　　0：正常終了
　　1：エラー
AH = エラーステータス
　　3：ビデオボード未実装
　　4：スチル/フレームバッファモードではない

機　能

ビデオボードのフレームバッファを高速にクリアする.

注　意

本ファンクションを実行するには，必ず「ビデオボード動作モードの設定/取得(AH = 91H)」でスチルもしくはフレームバッファモードに設定しなければならない.

$SGPCTL	AH=30H	GS

SGPの起動/停止

入　力

AH = 30H
AL = 起動指定
　　1:起動
　　2:停止

出　力

Carry = 設定ステータス
　　0:起動完了
　　1:エラー
AH = エラーステータス
　　1:SGPの機能が設定されていない
　　2:タイムアウト(ハードウェア障害)

機　能

SGP(Super Graphic Processor)の実行開始/動作中止を設定する.

$ASKSGP	AH=31H	GS

SGPの動作状態の取得

入　力

AH = 31H

出　力

Carry = 獲得ステータス
0:正常獲得
1:SGPの機能が設定されていない
AL = 動作状態
0:停止
1:動作中

機　能

SGP(Super Graphic Processor)の動作状態を取得する.

$INITVSE	AH=60H	GS

フレーム2/3の消去

入　力

AH = 60H

出　力

なし

機　能

ビデオ特殊効果のためにフレーム2/3を消去する.

$SET_VSE	AH=61H	GS

ビデオ特殊効果の設定

入　力

AH = 61H

AL = 特殊効果コマンド

00：カラー反転処理
01：部分カラー反転処理
02：左右鏡処理(右画面がベース)
03：左右鏡処理(左画面がベース)
04：上下鏡処理(上画面がベース)
05：上下鏡処理(下画面がベース)
06：左右反転処理
07：上下反転処理
08：部分左右反転処理
09：部分上下反転処理
0A：4分割画面処理
0B：9分割画面処理
0C：16分割画面処理
0D：25分割画面処理
0E：部分縮小処理
0F：モザイク処理
10：部分モザイク処理
11：球面化処理
12：残像化処理
13：格子抜き転送処理
14：ランダム転送処理
15：ランダム圧縮処理
16：台形転送処理(右側圧縮)
1E：ズーム転送処理
1F：ソフトフォーカス処理
20：千鳥状リバース処理
21：輝度1.5転送倍処理
22：輝度1/2転送処理
23：ゆらぎ転送処理
24：鏡反射処理
25：奇数行左右反転処理
26：0度回転処理
27：90度回転処理
28：180度回転処理
29：270度回転処理
2B：部分論理演算処理
2C：輪郭抽出処理
2D：輪郭強調処理
2E：高輝度部分減輝処理
2F：ノイズ追加処理
30：白黒化処理
31：白黒拡大縮小処理
32：マスク付き白黒化処理1
33：マスク付き白黒化処理2
34：256色処理
35：256色拡大縮小処理

17：台形転送処理（右側圧縮，左右反転）　　36：256階調白黒化処理
18：台形転送処理（左側圧縮）
19：台形転送処理（左側圧縮，左右反転）
1A：台形転送処理（下側圧縮）
1B：台形転送処理（下側圧縮，上下反転）
1C：台形転送処理（上側圧縮）
1D：台形転送処理（上側圧縮，上下反転）

ES：BX ＝ パラメータ格納アドレス（コマンドによっては不要）

・パラメータなしコマンド

00，02，03，04，05，06，07，11，12，13，24
ES：BXは意味をもたない

・機能パラメータのみをもつコマンド

0A，0B，0C，0D，0F

ES：BX 機能パラメータ （ワード）
モザイクの場合は，モザイクを形成する1ブロックの大きさを設定する．
分割表示の場合は，1部分転送を繰り返す回数を設定する．

・領域指定パラメータのみをもつコマンド

01, 08, 09, 0E, 14, 15, 20, 21, 22, 23, 25, 26, 27, 28, 29, 2C, 2D, 2E, 2F

ES : BX	X1(左上の水平座標)	(ワード)
	Y1(左上の垂直座標)	(ワード)
	X2(右下の水平座標)	(ワード)
	Y2(右下の垂直座標)	(ワード)

・領域指定パラメータを2つもつコマンド

1E

ES : BX	X1(転送元左上の水平座標)	(ワード)
	Y1(転送元左上の垂直座標)	(ワード)
	X2(転送元右下の水平座標)	(ワード)
	Y2(転送元右下の垂直座標)	(ワード)
	X3(転送先左上の水平座標)	(ワード)
	Y3(転送先左上の垂直座標)	(ワード)
	X4(転送先右下の水平座標)	(ワード)
	Y4(転送先右下の垂直座標)	(ワード)

・機能パラメータと領域指定パラメータをもつコマンド

10，16，17，18，19，1A，1B ，1C，1D，1F，2B，30，34

ES : BX	機能パラメータ	(ワード)
	X1(左上の水平座標)	(ワード)
	Y1(左上の垂直座標)	(ワード)
	X2(右下の水平座標)	(ワード)
	Y2(右下の垂直座標)	(ワード)

・機能パラメータとパターンアドレスをもつコマンド

2A，32，33

ES : BX	機能パラメータ	(ワード)
	パターン格納オフセットアドレス	(ワード)
	パターン格納セグメントアドレス	(ワード)

・機能パラメータと領域指定パラメータを2つもつコマンド

31，35，36

ES ： BX	機能パラメータ	(ワード) (*1)
	X1(転送元左上の水平座標)	(ワード)
	Y1(転送元左上の垂直座標)	(ワード)
	X2(転送元右下の水平座標)	(ワード)
	Y2(転送元右下の垂直座標)	(ワード)
	X3(転送先左上の水平座標)	(ワード)
	Y3(転送先左上の垂直座標)	(ワード)
	X4(転送先右下の水平座標)	(ワード)
	Y4(転送先右下の垂直座標)	(ワード)

(*1)35，36 ではバイトとなる.

出　力

Carry ＝ 処理ステータス

0 ： 正常設定

1 ： 設定不完全

機　能

ワーキングRAM上に特殊効果コマンドを展開する.

① SGPを停止させる.

② ワーキングRAMのアドレス0080H(A800 ： 0100)から特殊効果コマンドの展開を行う.

③ SGPを起動する.

$ASK_VSE	AH=62H	

ビデオ特殊効果状態の取得

入　力

AH = 62H

出　力

Carry = 状態ステータス
　　0 ： 特殊効果あり
　　1 ： 特殊効果なし
AL = 特殊効果(Carry=0のみ)

00：カラー反転処理
01：部分カラー反転処理
02：左右鏡処理(右画面がベース)
03：左右鏡処理(左画面がベース)
04：上下鏡処理(上画面がベース)
05：上下鏡処理(下画面がベース)
06：左右反転処理
07：上下反転処理
08：部分左右反転処理
09：部分上下反転処理
0A：4分割画面処理
0B：9分割画面処理
0C：16分割画面処理
0D：25分割画面処理
0E：部分縮小処理
0F：モザイク処理
10：部分モザイク処理
11：球面化処理
1E：ズーム転送処理
1F：ソフトフォーカス処理
20：千鳥状リバース処理
21：輝度1.5転送倍処理
22：輝度1/2転送処理
23：ゆらぎ転送処理
24：鏡反射処理
25：奇数行左右反転処理
26：0度回転処理
27：90度回転処理
28：180度回転処理
29：270度回転処理
2B：部分論理演算処理
2C：輪郭抽出処理
2D：輪郭強調処理
2E：高輝度部分減輝処理
2F：ノイズ追加処理
30：白黒化処理

12：残像化処理
13：格子抜き転送処理
14：ランダム転送処理
15：ランダム圧縮処理
16：台形転送処理(右側圧縮)
17：台形転送処理(右側圧縮，左右反転)
18：台形転送処理(左側圧縮)
19：台形転送処理(左側圧縮，左右反転)
1A：台形転送処理(下側圧縮)
1B：台形転送処理(下側圧縮，上下反転)
1C：台形転送処理(上側圧縮)
1D：台形転送処理(上側圧縮，上下反転)
31：白黒拡大縮小処理
32：マスク付き白黒化処理1
33：マスク付き白黒化処理2
34：256色処理
35：256色拡大縮小処理
36：256階調白黒化処理
FF：上記以外の特殊効果

機　能

現在のビデオ特殊効果の状態を取得する.

第4章

ジオメトリック系ファンクション

$INITGLO	AH=70H	
ジオメトリック機能の初期化		

入　力

AH = 70H
AL = 動作モード
　00H ： 直接実行モード
　80H ： バックグラウンド実行モード(GSのみ)
BX = 1000H(ペイントスタックのサイズ)

出　力

Carry = 初期化ステータス
　0 ： 初期化完了
　1 ： スタックサイズが異常

機　能

ジオメトリック動作モードおよびペイントのスタックサイズを設定する.

ペイントスタックはペイント時の座標バックアップおよび矩形描画でのパターン格納領域に用いられる.

$ASKGLO	AH=71H	

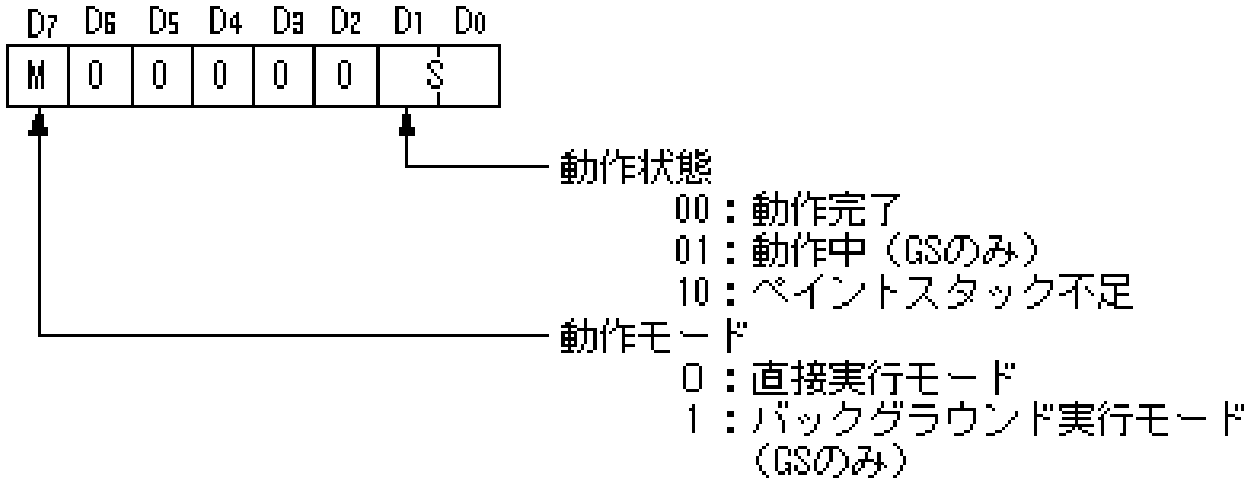
ジオメトリック動作状態の取得

入　力

AH = 71H

出　力

AL = 動作モード



BX = ペイント用スタックサイズ

DX = 現在実行中のコマンド(GSのみ)

SI = 現在実行中のコマンド格納ワードアドレス(GSのみ)

DI = バックグラウンド動作で使用しているワーキングRAMの上限
(GSのみ)

Carry = ステータス

0 ： ジオメトリック正常動作

1 ： =ジオメトリック動作状態ではない
（「ジオメトリック機能の初期化(AH=70H)」が行われていない)

機　能

ジオメトリック動作状態と動作モードを得る.

注　意

「閉領域の塗りつぶし[領域=境界色の内側](AH=7CH)」,「閉領域の塗りつぶし[領域=描画開始点と同一色](AH=7DH)」がスタックが足りないため終了した場合,本ファンクションは「ペイントスタック不足」を返す.

「ペイントスタック不足」は,他のジオメトリック系ファンクションを実行した時点でリセットされる.

$SELFRAME	AH=72H	

描画フレームの設定

入　力

AH = 72H

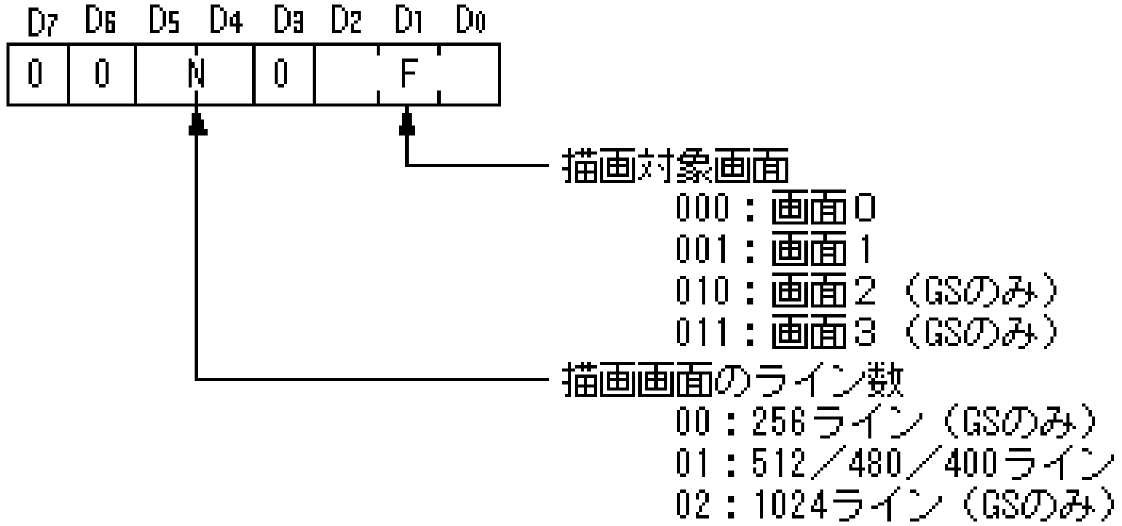
AL = 描画モード

1 ： 8ビット/ピクセル

2 ： 16ビット/ピクセル(GSのみ)

3 ： 24ビット/ピクセル(GSのみ)

BL = フレーム選択



出　力

Carry = 選択ステータス

0 ： 正常選択

1 ： 選択不能

機　能

ジオメトリック描画を行う画面を選択する.

したがって，ジオメトリック描画ファンクションを用いる場合は，あらかじめ本ファンクションを実行しておく必要がある.

画面選択(BL)における使用画面を以下に示す.

「入力条件」以外の設定は不能.

入力条件	画面0	画面1	画面2	画面3	
8　0　256	○				GSのみ
8　1　256		○			GSのみ
8　2　256			○		GSのみ
8　3　256				○	GSのみ
8　0　512	○				
8　2　512		○			
8　0　1024	○				GSのみ
16　0　256	○				GSのみ
16　2　256		○			GSのみ
16　0　512	○				GSのみ
24　0　256	○				GSのみ

- 第3項：描画画面ライン数(256,512/480/400,1024)
- 第2項：開始フレーム番号(0,1,2,3)
- 第1項：ビット／ピクセル(8,16,24)

$CLS	AH=73H	
グラフィックスフレームの0クリア		

入　力

AH = 73H

出　力

Carry = 動作ステータス
　　0 ： 正常
　　1 ： フレームの定義が行われていない.
　　　　(「描画フレームの設定(AH=72H)」未実行)

機　能

「描画フレームの設定(AH=72H)」 で定義されたフレームを高速にクリアする.

$VIEW	AH=74H	
描画領域の設定		

入　力

AH = 74H

ES:BS = 定義パラメータテーブルアドレス

出　力

Carry = 動作ステータス

0 ： 正常

1 ： フレームの定義が行われていない.

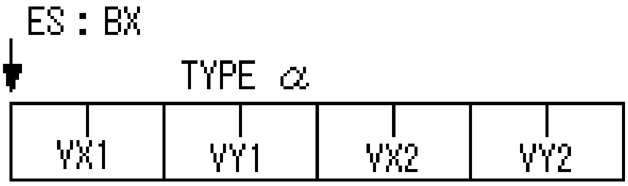
（「描画フレームの設定(AH=72H)」未実行)

パラメータ異常

機　能

ジオメトリック描画の描画領域を指定する.

定義パラメータテーブルの構成は以下のとおり.



VX1 ： 左端境界(0～1023/639)
VY1 ： 上端境界(0～1023/639)
VX2 ： 右端境界(0～1023/639)
VY2 ： 下端境界(0～1023/639)

以下の条件を満足しない場合は「パラメータ異常」となる.

・0 ≦ 左端境界 ≦ 右端境界 ＜ 1024/640
・0 ≦ 上端境界 ≦ 下端境界 ＜ フレームライン数
(256/400/480/512/1024)

$PSET	AH=75H	

点の描画

入　力

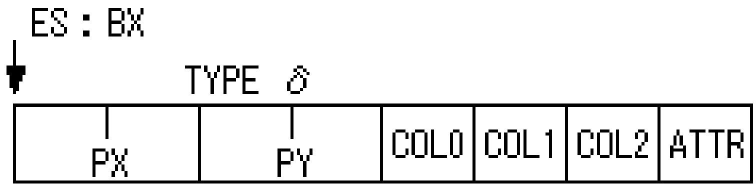
AH = 75H
ES:BX = 描画パラメータテーブルアドレス

出　力

Carry = 描画ステータス
0 ： 正常終了
1 ： フレームの定義が行われていない．
（「描画フレームの設定(AH=72H)」未実行)
パラメータ異常

機　能

描画パラメータテーブルの構成は以下のとおり.



PX ： 点の水平座標（-16384～16383）
PY ： 点の垂直座標（-16384～16383）
COL0 ： カラーコード
　256色時 ： カラーコード
　65,000色時 ： カラー値の下位8ビット(GSのみ)
　1677万色時 ： 青レベル(GSのみ)
COL1 ： カラーコード
　256色時 ： 未使用
　65,000色時 ： カラー値の上位8ビット(GSのみ)
　1677万色時 ： 赤レベル(GSのみ)
COL2 ： カラーコード
　256色時 ： 未使用
　65,000色時 ： 未使用(GSのみ)
　1677万色時 ： 緑レベル(GSのみ)
ATTR ： 論理演算
　0 ： SET 通常描画
　1 ： NOT 反転描画
　2 ： AND 画面との論理積
　3 ： OR 画面との論理和
　4 ： XOR 画面との排他的論理和

$STYLE	AH=76H	

ラインスタイルの設定

入　力

AH = 76H
BX = ラインスタイルパターン

出　力

Carry = 動作ステータス
　　0 ： 正常
　　1 ： フレームの定義が行われていない.
　　　（「描画フレームの設定(AH=72H)」未実行)

機　能

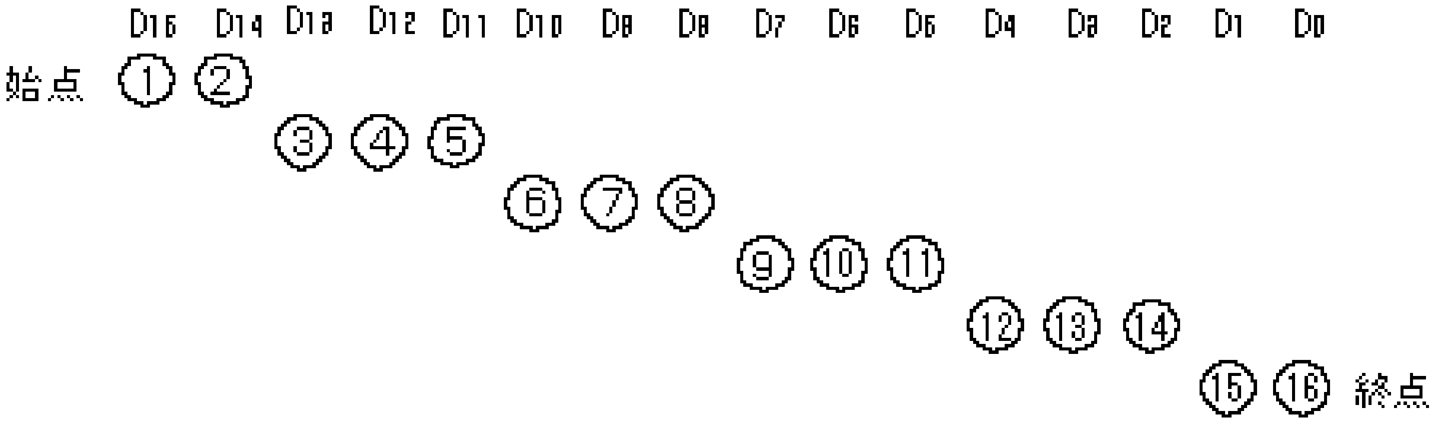
「線分の描画(AH=77H)」，「線分の連続描画(AH=78H)」，「箱の描画(AH=79H)」，「箱の塗りつぶし描画(AH=7AH)」に有効線分を描画する際参照するラインスタイルパターンを定義する.

ラインスタイルは次のようなビット構成を成す.

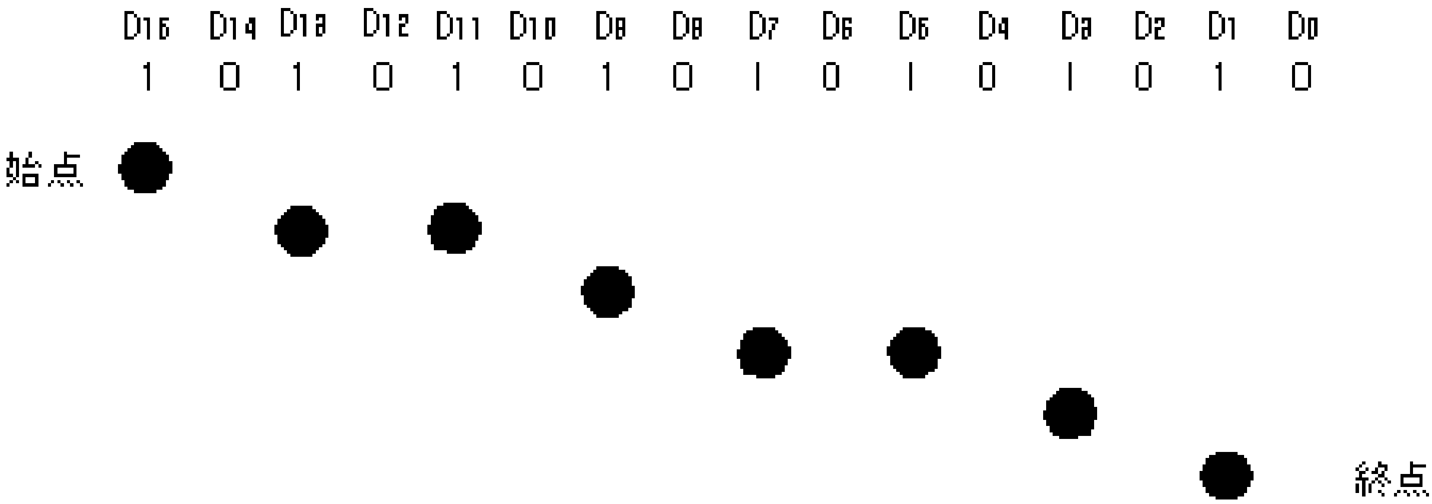
	D_{15}	D_{14}	D_{13}	D_{12}	D_{11}	D_{10}	D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
始点	①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩	⑪	⑫	⑬	⑭	⑮	⑯	終点

D_{15}から順に使用される.

描画時は以下のように使用される.



各ラインスタイルが次の場合，直線描画は以下のとおりになる.



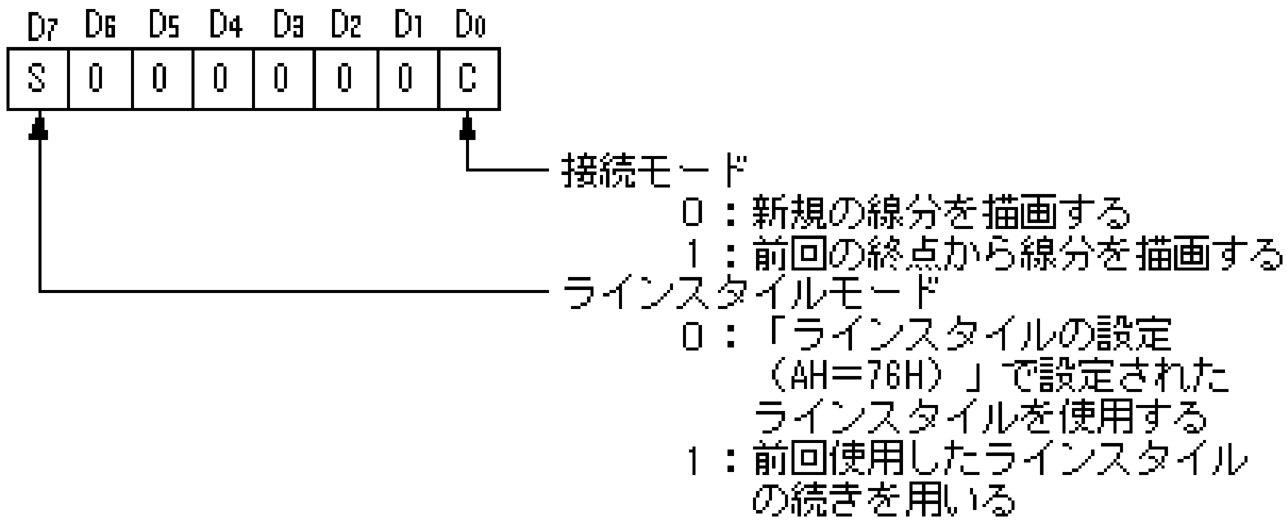
$LINE	AH=77H	

線分の描画

入　力

AH = 77H

AL = 描画モード



ES:BX = 描画パラメータテーブルアドレス

出　力

Carry = 描画ステータス

0 ： 正常終了

1 ： フレームの定義が行われていない.

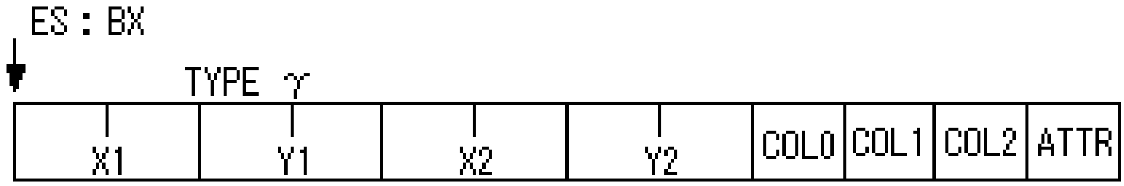
（「描画フレームの設定(AH=72H)」未実行)

パラメータ異常

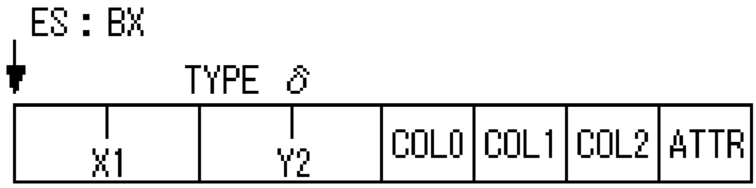
機　能

描画パラメータテーブルの構成は以下のとおり.

・接続モード = 0　の場合



・接続モード = 1　の場合



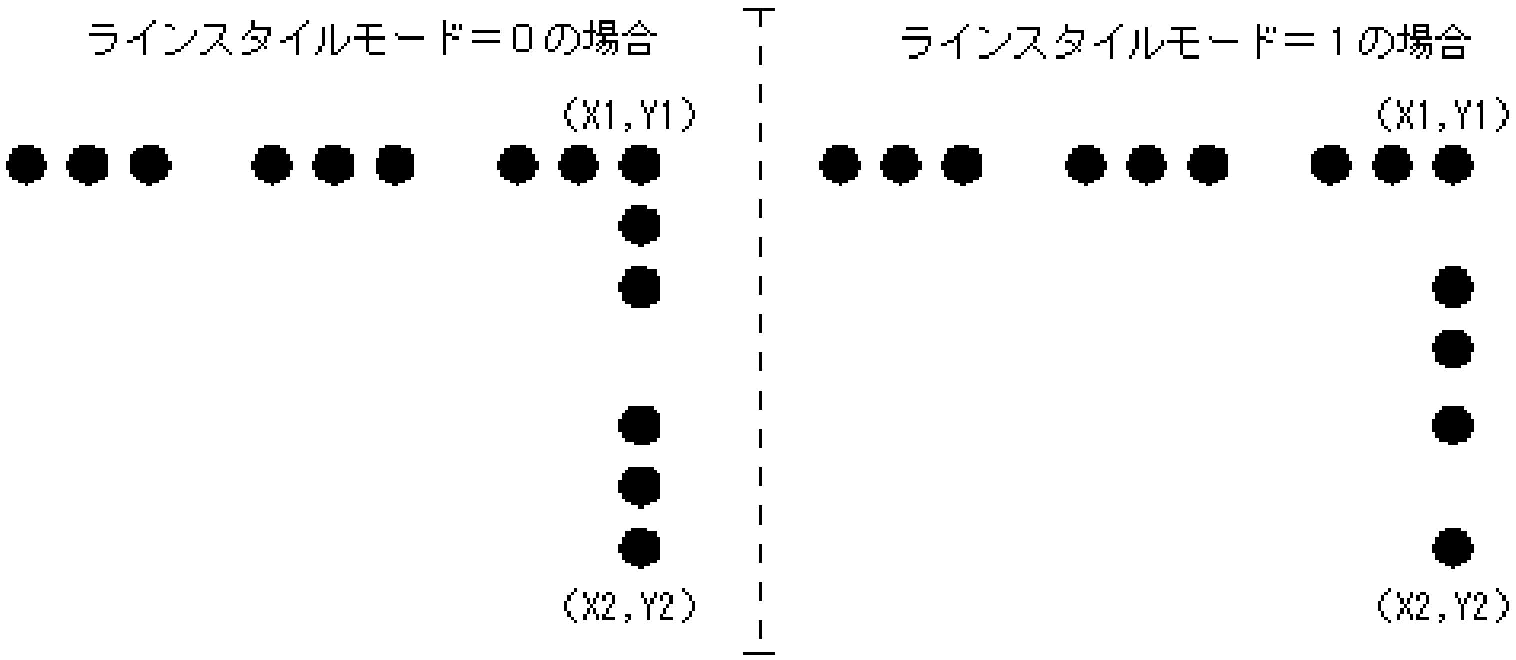
X1　：　開始点水平座標(−16384～16383)
Y1　：　開始点垂直座標(−16384～16383)
X2　：　終了点水平座標(−16384～16383)
Y2　：　終了点垂直座標(−16384～16383)

COL0　：　カラーコード
　　256色時　：　カラーコード
　　65,000色時　：　カラー値の下位8ビット(GSのみ)
　　1677万色時　：　青レベル(GSのみ)
COL1　：　カラーコード
　　256色時　：　未使用 → 0
　　65,000色時　：　カラー値の上位8ビット(GSのみ)
　　1677万色時　：　赤レベル(GSのみ)

COL2 ： カラーコード
　　256色時 ： 未使用 → 0
　　65,000色時 ： 未使用 → 0 (GSのみ)
　　1677万色時 ： 緑レベル(GSのみ)
ATTR ： 論理演算
　　0 ： SET 通常描画
　　1 ： NOT 反転描画
　　2 ： AND 画面との論理積
　　3 ： OR 画面との論理和
　　4 ： XOR 画面との排他的論理和

ラインスタイルモードにより複数の線分に連続してラインスタイルパターンを利用することができる.

・ラインスタイル(1110111011101110)の場合



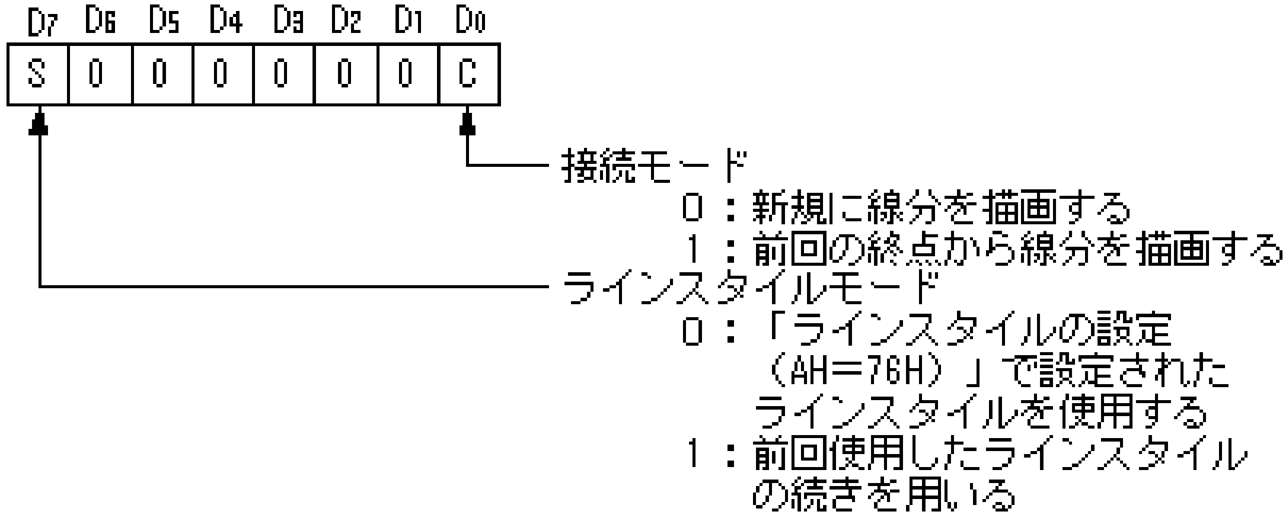
$CONNECT	AH=78H	

線分の連続描画

入　力

AH = 78H

AL = 描画モード



ES:BX = 描画パラメータテーブルアドレス

CX = 線分の数

出　力

Carry = 描画ステータス

0 ： 正常終了

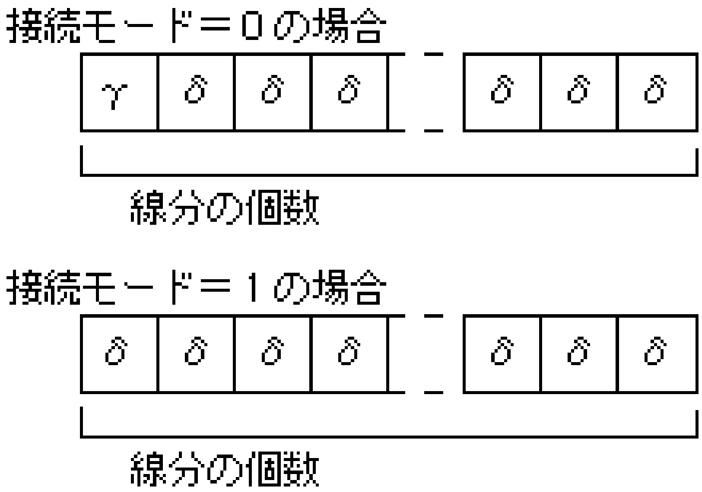
1 ： フレームの定義が行われていない.

（「描画フレームの設定(AH=72H)」未実行)

パラメータ異常

機　能

描画パラメータテーブルの構成は以下のとおり.



γ　は線分の描画(AH=77H)コマンドでのTYPE γ の形式を示す.

δ　は線分の描画(AH=77H)コマンドでのTYPE δ の形式を示す.

$BOX	AH=79H	

箱の描画

入　力

AH = 79H

ES:BX = 描画パラメータテーブルアドレス

出　力

Carry = 描画ステータス

0 ： 正常終了

1 ： フレームの定義が行われていない.

（「描画フレームの設定(AH=72H)」未実行)

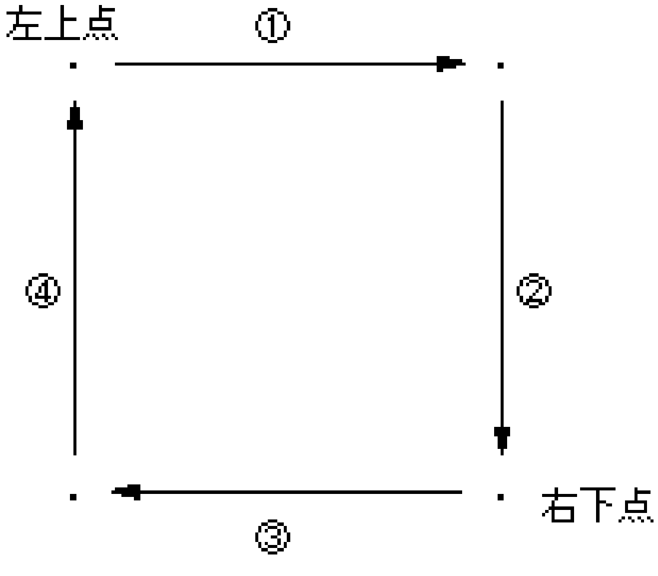
パラメータ異常

機　能

左上点, 右下点を対角とした箱(四角形)を描画する.

描画パラメータテーブルの構成は「線分の描画(AH=77H)」コマンドのTYPE γと同様.

描画は以下の順で行われ, ラインスタイルパターンが用いられる.



$BOXFILL	AH=7AH	

箱の塗りつぶし描画

入　力

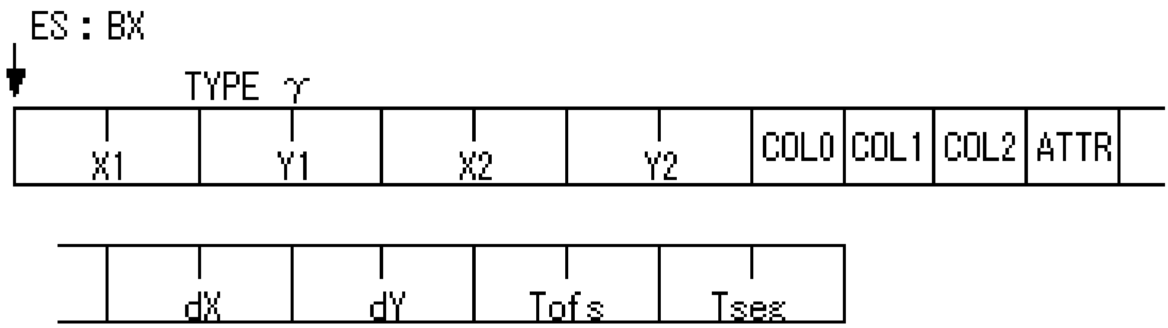
AH = 7AH
ES:BX = 描画パラメータテーブルセグメントアドレス

出　力

Carry = 描画ステータス
　　0 ： 正常終了
　　1 ： フレームの定義が行われていない.
　　　　(「描画フレームの設定(AH=72H)」未実行)
　　　　パラメータ異常

機　能

左上点，右下点を対角とした箱(四角形)を塗りつぶし描画する.
描画パラメータテーブルの構成は以下のとおり.



X1 ： 開始点水平座標(−16384～16383)
Y1 ： 開始点垂直座標(−16384～16383)
X2 ： 終了点水平座標(−16384～16383)

Y2 ： 終了点垂直座標(-16384～16383)

COL0 ： カラーコード

256色時 ： カラーコード

65,000色時 ： カラー値の下位8ビット(GSのみ)

1677万色時 ： 青レベル(GSのみ)

COL1 ： カラーコード

256色時 ： 未使用 → 0

65,000色時 ： カラー値の上位8ビット(GSのみ)

1677万色時 ： 赤レベル(GSのみ)

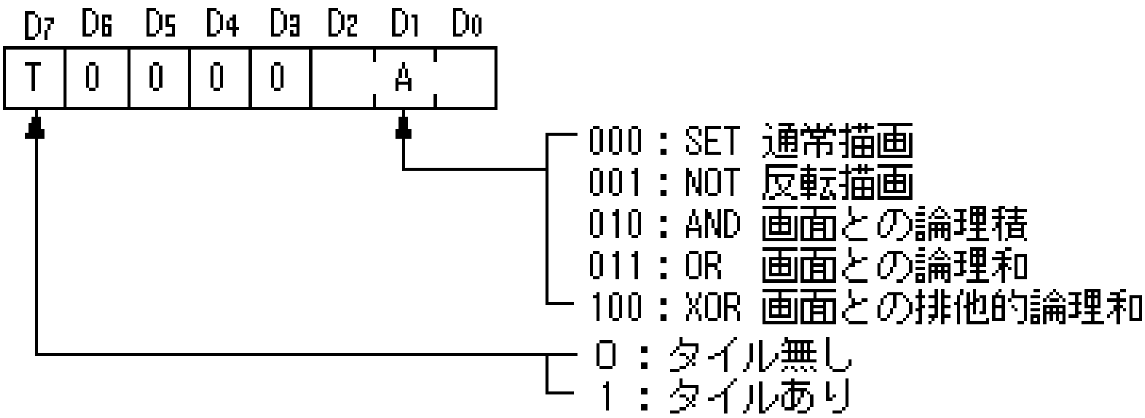
COL2 ： カラーコード

256色時 ： 未使用 → 0

65,000色時 ： 未使用 → 0 (GSのみ)

1677万色時 ： 緑レベル(GSのみ)

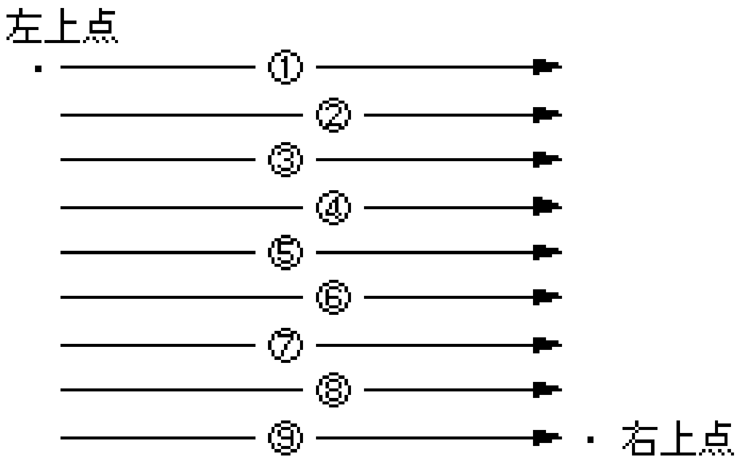
ATTR ： 論理演算



dX・dY ： タイルパターンの水平・垂直サイズ

Tofs・Tseg ： タイルパターンの格納オフセット・セグメントアドレス

タイルの指定がない場合，描画は以下の順で行われラインスタイルパターンが用いられる.



　タイルの指定がある場合，描画にはタイルパターンが用いられる．
　タイルパターンの格納形式は「矩形パターンの描画(AH=80H)」を参照のこと．

$CIRCLE	AH=7BH	

円と円弧の描画

入　力

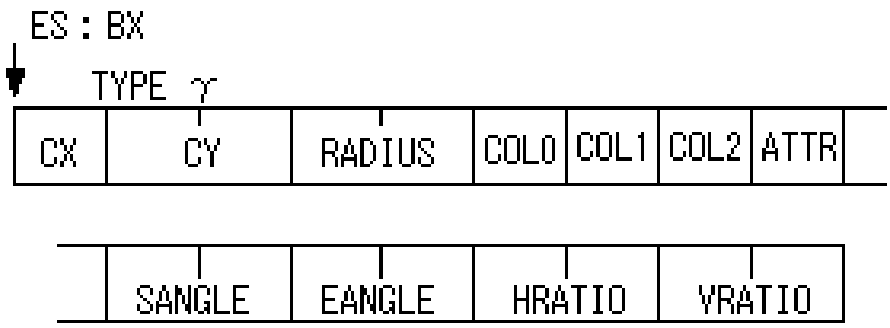
AH = 7BH
ES:BX = 描画パラメータテーブルアドレス

出　力

Carry = 描画ステータス
　　0 ： 正常終了
　　1 ： フレームの定義が行われていない.
　　　（「描画フレームの設定(AH=72H)」未実行)
　　　パラメータ異常

機　能

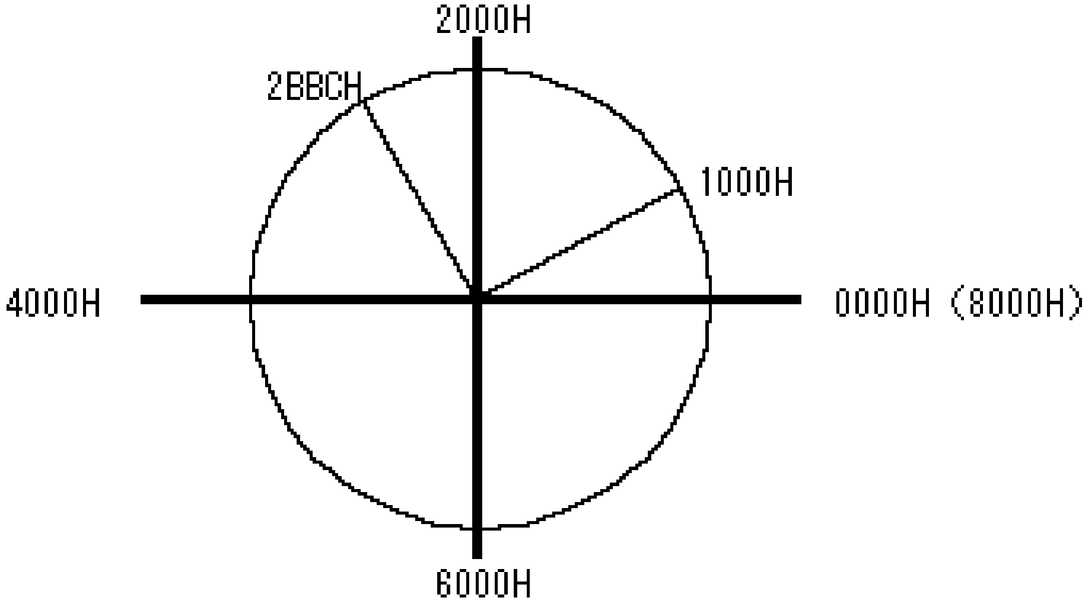
中心から半径分はなれた円もしくは円弧を描画する.
描画パラメータテーブルの構成は以下のとおり.



CX ： 円の中心水平座標(−16384〜16383)
CY ： 円の中心垂直座標(−16384〜16383)
RADIUS ： 円の半径(1〜16383)

COL0 ： カラーコード
　256色時 ： カラーコード
　65,000色時 ： カラー値の下位8ビット(GSのみ)
　1677万色時 ： 青レベル(GSのみ)
COL1 ： カラーコード
　256色時 ： 未使用 → 0
　65,000色時 ： カラー値の上位8ビット(GSのみ)
　1677万色時 ： 赤レベル(GSのみ)
COL2 ： カラーコード
　256色時 ： 未使用 → 0
　65,000色時 ： 未使用 → 0 (GSのみ)
　1677万色時 ： 緑レベル(GSのみ)
ATTR ： 論理演算
　0 ： SET 通常描画
　1 ： NOT 反転描画
　2 ： AND 画面との論理積
　3 ： OR 画面との論理和
　4 ： XOR 画面との排他的論理和
SANGLE ： 開始角度(360° = 8000H)
EANGLE ： 終了角度(360° = 8000H)
HRATIO ： 水平方向の扁平率(真円では0)
VRATIO ： 垂直方向の扁平率(真円では0)

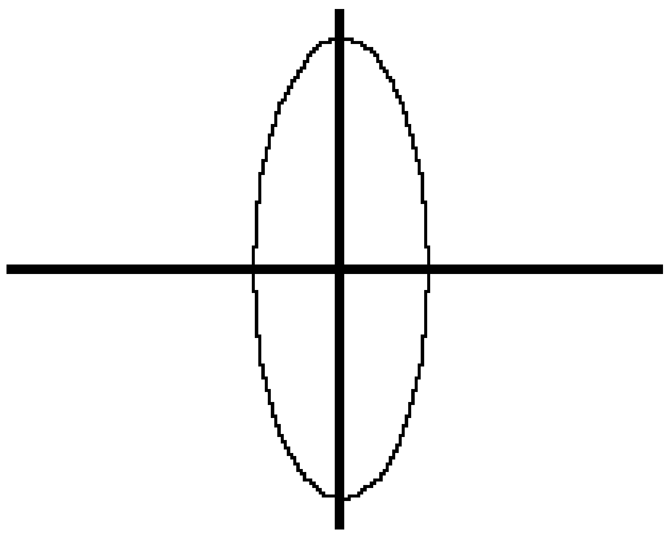
・　角度の概念を示す.



例	0°	→	0000H
	45°	→	1000H
	90°	→	2000H
	123°	→	2BBCH
	360°	→	8000H

・　扁平率の概念を示す.

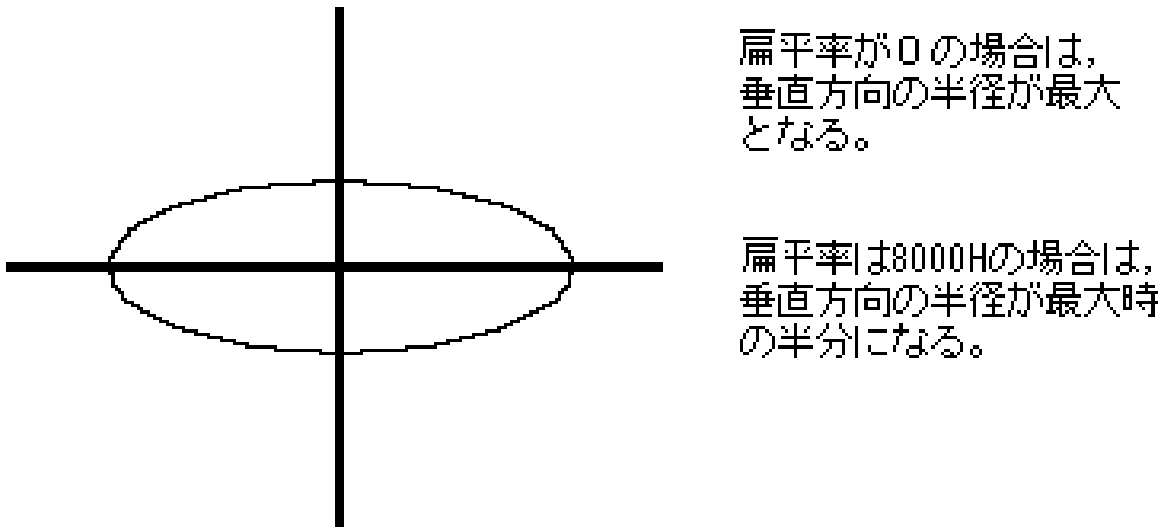
扁平率が8000Hの場合は，水平方向の半径が最大時の半分になる.



扁平率が０の場合は，
水平方向の半径が最大
となる。

扁平率は8000Hの場合は，
水平方向の半径が最大時
の半分になる。

垂直方向の扁平率が大きいほど円は横長になる.



注意 ： 水平/垂直扁平率はどちらか一方は必ず0でなければならない.扁平率の指定は内部で17ビットに拡張される.

例 ： 0000H → 10000H(100%)
1234H → 01234H(71.106%)
8000H → 08000H(50%)
FFFFH → 0FFFFH(99.9985%)

$PAINT1	AH=7CH	

閉領域の塗りつぶし［領域=境界色の内側］

入　力

AH = 7CH
ES:BX = 描画パラメータテーブルアドレス

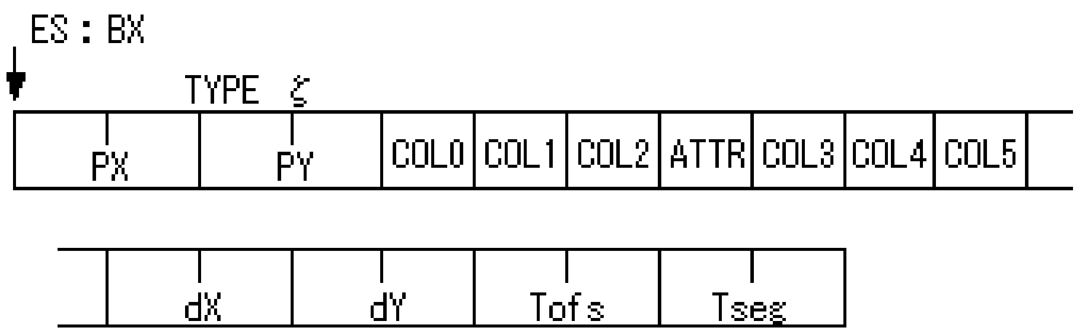
出　力

Carry = 描画ステータス
　0 ： 正常終了
　1 ： フレームの定義が行われていない.
　　（「描画フレームの設定(AH=72H)」未実行）
　　パラメータ異常

機　能

ペイント開始座標で指定された位置から境界色を左右上下に探し，その閉領域を描画色で塗りつぶす.

描画パラメータテーブルの構成は以下のとおり.



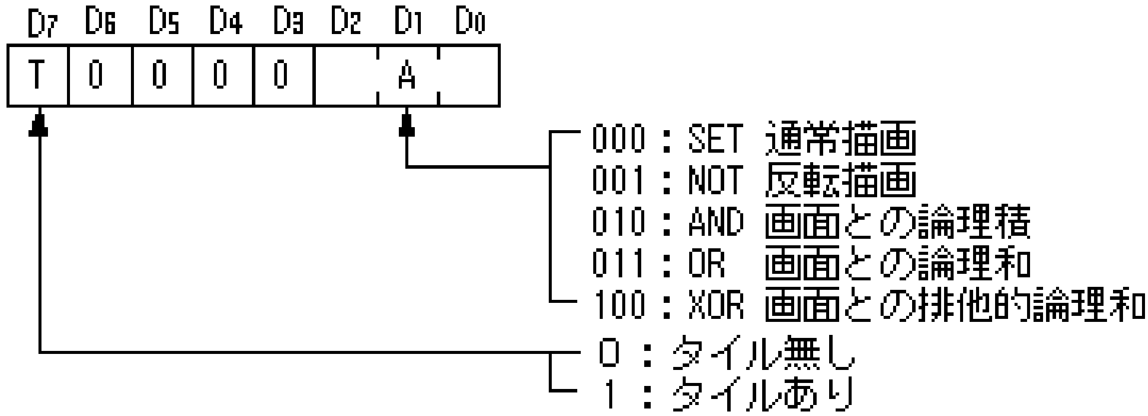
PX ： ペイント開始水平座標(描画可能領域)
PY ： ペイント開始垂直座標(描画可能領域)

COL0 ： 描画色
　256色時 ： カラーコード
　65,000色時 ： カラー値の下位8ビット(GSのみ)
　1677万色時 ： 青レベル(GSのみ)

COL1 ： 描画色
　256色時 ： 未使用 → 0
　65,000色時 ： カラー値の上位8ビット(GSのみ)
　1677万色時 ： 赤レベル(GSのみ)

COL2 ： 描画色
　256色時 ： 未使用 → 0
　65,000色時 ： 未使用(GSのみ)
　1677万色時 ： 緑レベル(GSのみ)

ATTR ： 論理演算



COL3 ： 境界色
　256色時 ： カラーコード
　65,000色時 ： カラー値の下位8ビット(GSのみ)
　1677万色時 ： 青レベル(GSのみ)

COL4 :境界色
　256色時 ： 未使用 → 0
　65,000色時 ： カラー値の上位8ビット(GSのみ)
　1677万色時 ： 赤レベル(GSのみ)

COL5 :境界色
256色時 : 未使用 → 0
65,000色時 : 未使用 → 0 (GSのみ)
1677万色時 : 緑レベル(GSのみ)
dX : タイルパターンの水平サイズ
dY : タイルパターンの垂直サイズ
Tofs : タイルパターンの格納オフセットアドレス
Tseg : タイルパターンの格納セグメントアドレス

タイルの指定があった場合は，COL0を無視し，タイルパターンで閉領域を塗りつぶす.

タイルパターンの格納形式は「矩形パターンの描画(AH=80H)」を参照のこと.

$PAINT2	AH=7DH	

閉領域の塗りつぶし
［領域=描画開始点と同一色］

入　力

AH = 7DH

ES:BX = 描画パラメータテーブルアドレス

出　力

Carry = 描画ステータス

0 ： 正常終了

1 ： フレームの定義が行われていない.

（「描画フレームの設定(AH=72H)」未実行）

パラメータ異常

機　能

ペイント開始座標で指定された位置からペイント開始点と同一色でない点を左右上下に探し，その閉領域を描画色で塗りつぶす.

描画パラメータテーブルの構成は以下のとおり.

ES：BX

TYPE η

PX	PY	COL0	COL1	COL2	ATTR	dX	dY	Tofs	Tseg

PX ： ペイント開始水平座標(描画可能領域)

PY ： ペイント開始垂直座標(描画可能領域)

COL0 ：描画色

256色時 ： カラーコード

65,000色時 ： カラー値の下位8ビット(GSのみ)

1677万色時 ： 青レベル(GSのみ)

COL1 ：描画色

256色時 ： 未使用 → 0

65,000色時 ： カラー値の上位8ビット(GSのみ)

1677万色時 ： 赤レベル(GSのみ)

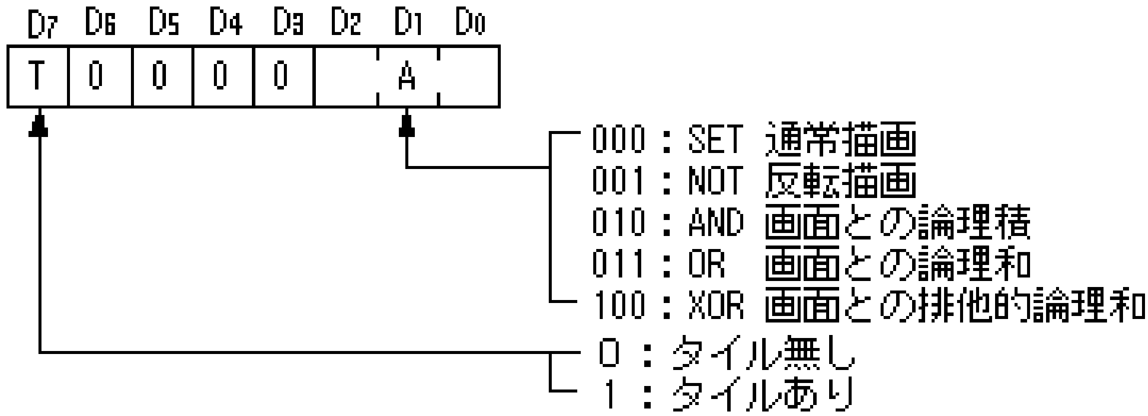
COL2 ：描画色

256色時 ： 未使用 → 0

65,000色時 ： 未使用 → 0 (GSのみ)

1677万色時 ： 緑レベル(GSのみ)

ATTR ： 論理演算



dX ： タイルパターンの水平サイズ

dY ： タイルパターンの垂直サイズ

Tofs ： タイルパターンの格納オフセットアドレス

Tseg ： タイルパターンの格納セグメントアドレス

タイルの指定があった場合は，COL0を無視しタイルパターンで閉領域を塗りつぶす．

タイルパターンの格納形式は「矩形パターンの描画(AH=80H)」を参照のこと．

$POINT	AH=7EH	

画面上の点の取得

入　力

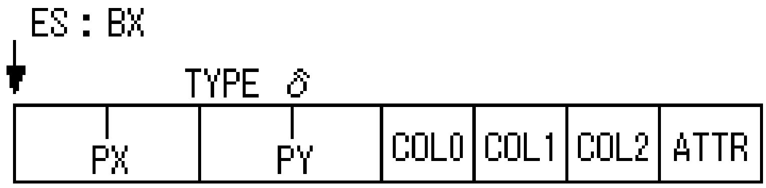
AH = 7EH
ES:BX = 描画パラメータテーブルアドレス

出　力

Carry = 描画ステータス
0 ： 正常終了
1 ： フレームの定義が行われていない.
（「描画フレームの設定(AH=72H)」未実行)
パラメータ異常

機　能

描画パラメータテーブルの構成は以下のとおり.



PX ： 点の水平座標(-16384～16383)
PY ： 点の垂直座標(-16384～16383)
COL0 ：描画色
　256色時 ： カラーコード
　65,000色時 ： カラー値の下位8ビット(GSのみ)
　1677万色時 ： 青レベル(GSのみ)
COL1 ：描画色
　256色時 ： 未使用 → 0
　65,000色時 ： カラー値の上位8ビット(GSのみ)
　1677万色時 ： 赤レベル(GSのみ)
COL2 ：描画色
　256色時 ： 未使用 → 0
　65,000色時 ： 未使用 → 0 (GSのみ)
　1677万色時 ： 緑レベル(GSのみ)
ATTR ： 論理演算

画面上の点を取得し，COL0と演算処理を行いCOL0へ結果を返す.

$PUTPAT	AH=80H	

矩形パターンの描画

入　力

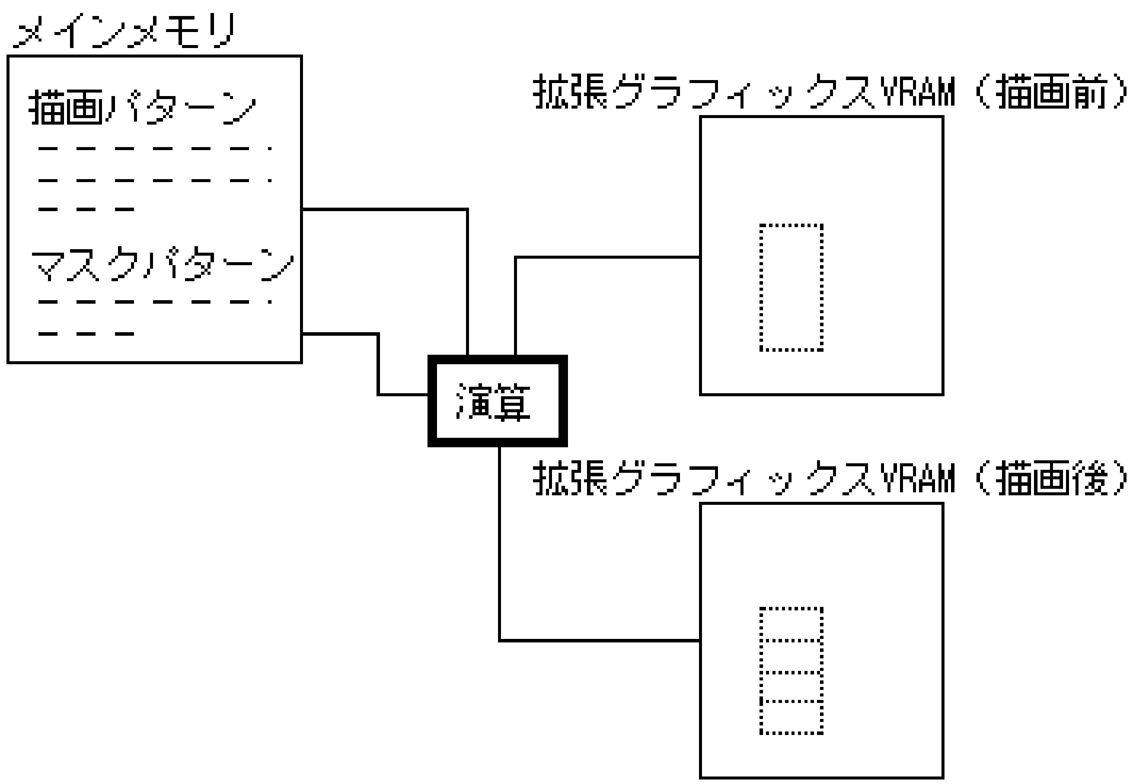
AH = 80H
ES:BX = 描画パラメータテーブルアドレス

出　力

Carry = 描画ステータス
0 ： 正常終了
1 ： フレームの定義が行われていない.
（「描画フレームの設定(AH=72H)」未実行)
パラメータ異常

機　能

以下のように転送処理を行う.



・ 描画パラメータテーブルの構成は以下のとおり.

ES：BX

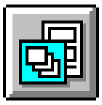
TYPE 0

dX	dY	Mofs	Mseg	Sofs	Sseg	X	Y	ATTR

dX ： 描画水平サイズ(描画可能サイズ)
dY ： 描画垂直サイズ(描画可能サイズ)
Mofs ： マスクパターン格納オフセットアドレス
Mseg ： マスクパターン格納セグメントアドレス
Sofs ： 描画パターン格納オフセットアドレス
Sseg ： 描画パターン格納セグメントアドレス
X ： 描画水平座標(描画可能領域)
Y ： 描画垂直座標(描画可能領域)
ATTR ： 論理演算
　0 ： SET 通常描画
　1 ： NOT 反転描画
　2 ： AND 画面との論理積
　3 ： OR 画面との論理和
　4 ： XOR 画面との排他的論理和

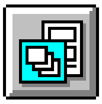
- マスクパターンの格納形式は以下のとおり.

▼*マスクパターンの格納形式(INT 0D8H/80H)*

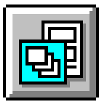


- 描画パターンの格納形式は以下のとおり.

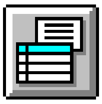
▼*描画パターンの格納形式(8ビット/ピクセルモード)*



▼*描画パターンの格納形式(16ビット/ピクセルモード)*



▼*描画パターンの格納形式(24ビット/ピクセルモード)[GSのみ]*



注　意

矩形描画系コマンド(「矩形パターンの描画(AH=80H)」，「矩形パターンの取得(AH=81H)」，「矩形パターンの転送(AH=82H)」)およびペイント系コマンド(「閉領域の塗りつぶし[領域=境界色の内側](AH=7CH)」，「閉領域の塗りつぶし[領域=描画開始点と同一色](AH=7DH)」)をバックグラウンド動作中に本コマンドを2回以上指定した場合，バックグラウンド動作すべてを終了するのを待って機能する.

「描画領域の設定(AH=74H)」による描画領域指定は無視される.

大きな矩形を描画する場合は，あらかじめ「ジオメトリック機能の初期化(AH=70H)」で大きめのペイントスタックサイズを確保する必要がある.

$GETPAT	AH=81H	

矩形パターンの取得

入　力

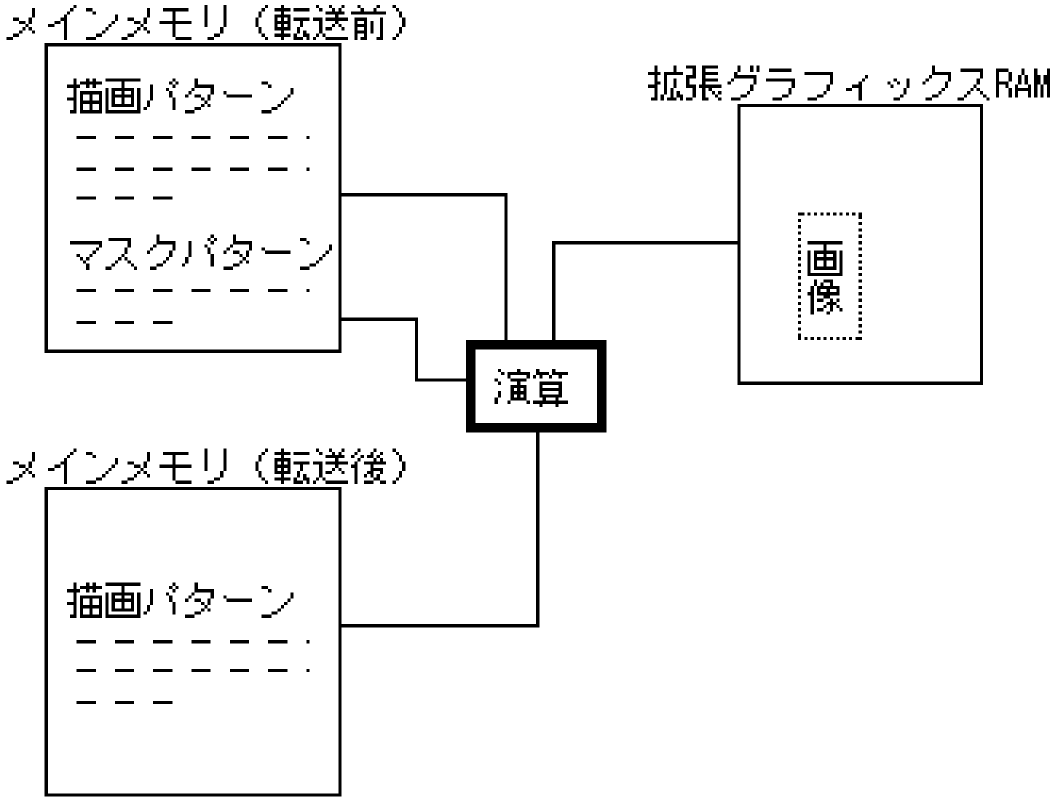
AH = 81H
ES:BX = 転送パラメータテーブルアドレス

出　力

Carry = 転送ステータス
　0 ： 正常終了
　1 ： フレームの定義が行われていない.
　　（「描画フレームの設定(AH=72H)」未実行)
　　パラメータ異常

機　能

以下のように転送処理を行う.



転送パラメータテーブルの構成は以下のとおり.

ES：BX

TYPE ¿

dX	dY	Mofs	Mseg	X	Y	Dofs	Dseg	ATTR

dX ： 転送水平サイズ(転送可能サイズ)
dY ： 転送垂直サイズ(転送可能サイズ)
Mofs ： マスクパターン格納オフセットアドレス
Mseg ： マスクパターン格納セグメントアドレス
X ： 転送水平座標(転送可能領域)
Y ： 転送垂直座標(転送可能領域)
Dofs ： 転送パターン格納オフセットアドレス
Dseg ： 転送パターン格納セグメントアドレス
ATTR ： 論理演算
　0 ： SET 通常転送
　1 ： NOT 反転転送
　2 ： AND 画面との論理積
　3 ： OR 画面との論理和
　4 ： XOR 画面との排他的論理和

マスクパターン/画像パターンの格納形式は「矩形パターンの描画(AH=80H)」と同様.

注　意

矩形描画系コマンド(「矩形パターンの描画(AH=80H)」，「矩形パターンの取得(AH=81H)」，「矩形パターンの転送(AH=82H)」)およびペイント系コマンド(「閉領域の塗りつぶし[領域=境界色の内側](AH=7CH)」，「閉領域の塗りつぶし[領域=描画開始点と同一色](AH=7DH)」)をバックグラウンド動作中に本コマンドを2回以上指定した場合，バックグラウンド動作すべてを終了するのを待って機能する(GSのみ).

「描画領域の設定(AH=74H)」による描画領域指定は無視される.

大きな矩形を描画する場合は，あらかじめ「ジオメトリック機能の初期化(AH=70H)」で大きめのペイントスタックサイズを確保する必要がある(GSのみ).

$MOVPAT	AH=82H	
矩形パターンの転送		

入　力

AH = 82H

ES:BX = 転送パラメータテーブルアドレス

出　力

Carry = 転送ステータス

0 ： 正常終了

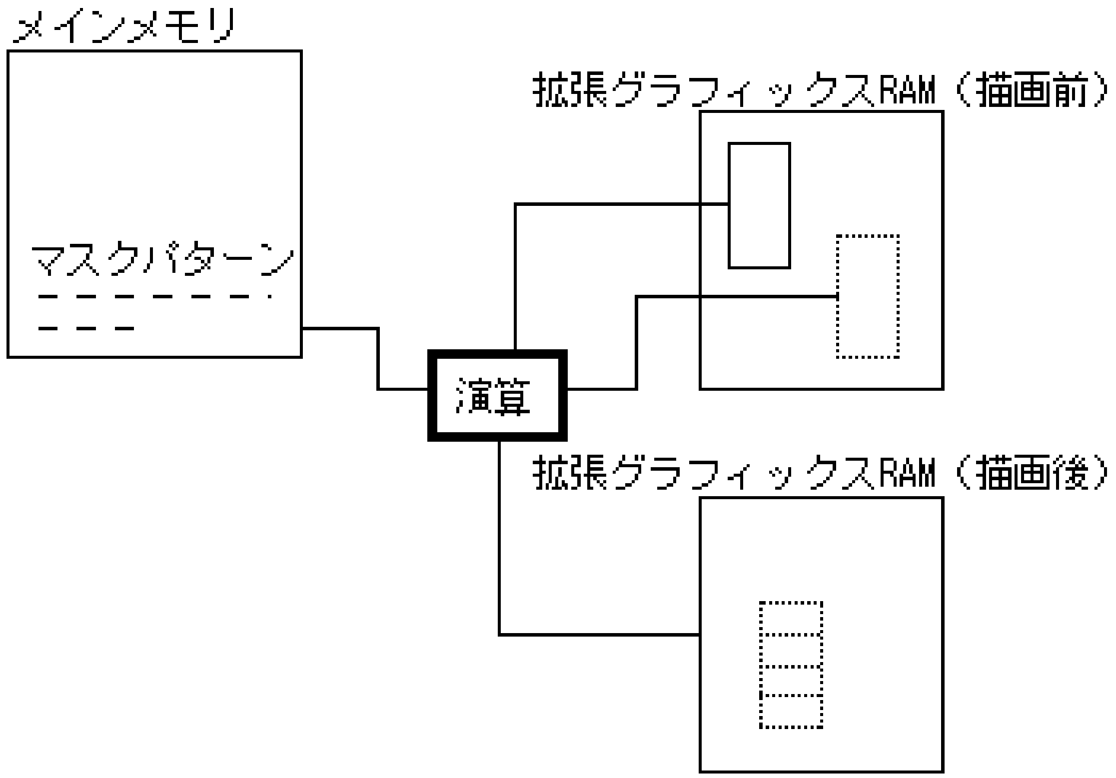
1 ： フレームの定義が行われていない.

（「描画フレームの設定(AH=72H)」未実行)

パラメータ異常

機　能

以下のように転送処理を行う.



・ 転送パラメータテーブルの構成は以下のとおり.

ES：BX

TYPE κ

dX	dY	Mofs	Mseg	SX	SY	DX	DY	ATTR

dX ： 転送水平サイズ(転送可能サイズ)
dY ： 転送垂直サイズ(転送可能サイズ)
Mofs ： マスクパターン格納オフセットアドレス
Mseg ： マスクパターン格納セグメントアドレス
SX ： 転送元水平座標(転送可能領域)
SY ： 転送元垂直座標(転送可能領域)
DX ： 転送先水平座標(転送可能領域)
DY ： 転送先垂直座標(転送可能領域)
ATTR ： 論理演算
　0 ： SET 通常転送
　1 ： NOT 反転転送
　2 ： AND 画面との論理積
　3 ： OR 画面との論理和
　4 ： XOR 画面との排他的論理和

マスクパターン/画像パターンの格納形式は「矩形パターンの描画(AH=80H)」と同様.

注　意

矩形描画系コマンド(「矩形パターンの描画(AH=80H)」，「矩形パターンの取得(AH=81H)」，「矩形パターンの転送(AH=82H)」およびペイント系コマンド(「閉領域の塗りつぶし[領域=境界色の内側](AH=7CH)」，「閉領域の塗りつぶし[領域=描画開始点と同一色](AH=7DH)」)をバックグラウンド動作中に本コマンドを2回以上指定した場合，バックグラウンド動作すべてを終了するのを待って機能する．(GSのみ)

「描画領域の設定(AH=74H)」コマンドによる描画領域指定は無視される．

大きな矩形を描画する場合は，あらかじめ「ジオメトリック機能の初期化(AH=70H)」で大きめのペイントスタックサイズを確保する必要がある．(GSのみ)

第3部
拡張サウンド
ドライバ

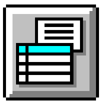
第1章

拡張サウンドドライバ
サポートファンクション

　以下のファンクションをサポートする(PC-98GS/9821/9801-73/9801-86いずれの環境下でも利用可能)．

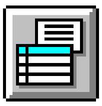
●YM2203互換BIOS機能一覧(INT 0DAH)

▼YM2203互換BIOSファンクション一覧



●拡張サウンドBIOS機能一覧(INT 0DAH)

▼拡張サウンドBIOSファンクション一覧



第2章

拡張サウンドドライバ概要

　本ドライバは，YM2608(OPNA)を利用して，FM3和音+SSG3和音またはFM6和音+SSG3和音+リズムによる音楽演奏機能を提供する．
　FM3和音+SSG3和音で利用する場合には，従来のサウンドBIOSと互換性を有する．

2.1　互換モードと拡張モード

本ドライバは互換モードと拡張モードの2つの動作モードをもつ．互換モードは従来のサウンドBIOSと互換性のあるモードであり，FM3和音+SSG3和音を利用して演奏することができる．

拡張モードはYM2203の上位音源ICであるYM2608用に新たに導入されたモードで，最大FM6和音+SSG3和音+リズムを利用して演奏することができる．

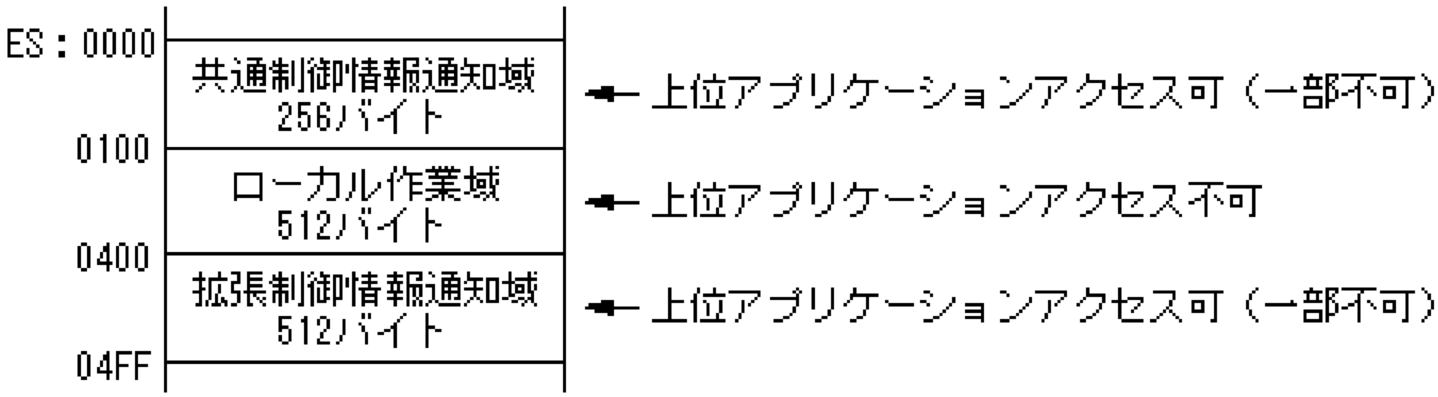
「ドライバの初期化(AH=00H)」で始めた場合は互換モードとして，「拡張モードでのドライバの初期化(AH=03H)」で始めた場合には拡張モードとして動作する．他のファンクションは2つのモードで共通に利用できるが，モードにより渡すパラメータが異なる場合がある．

2.2　リズム機能

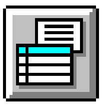
リズムパートの演奏は，あらかじめリズムパターンを用意しておき，ディレイドコマンドでは演奏するパターン番号を指定するという方法を用いる．

2.3　サウンドドライバ作業域

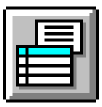
- システム共通域(0000 ： 05E0H) 4バイト
- 共通制御情報通知域 256バイト
- ローカル作業域 512バイト
- 拡張制御情報通知域 512バイト
- スタック 256バイト
- その他



▼*共通制御情報通知域詳細*



▼*チャネル制御用ワークエリア*



▼*拡張制御情報通知域詳細*

■リズムパターンバッファ詳細

0000H ： パターン0のオフセットアドレス
0002H ： パターン1のオフセットアドレス
00FCH ： パターン126のオフセットアドレス
00FEH ： パターン127のオフセットアドレス
0100H ： パターンデータ

+0	パターンデータ列のサイズ(+2～のサイズ，バイト)
+1	音長(1～48 ： 48/12step単位)
+2～	パターンデータ列

第3章

YM2203互換ファンクション

INITIALIZE	AH=00H	
ドライバの初期化		

入　力

AH = 00H
ES = サウンドドライバ作業域セグメントアドレス
プレイバッファ情報6チャネル分
BUF_OFF_n ： チャネルnのプレイバッファオフセット
BUF_SEG_n ： チャネルnのプレイバッファセグメント
BUF_LNG_n ： チャネルnのプレイバッファサイズ(バイト)

出　力

なし

機　能

　サウンドドライバを初期化する.

　サウンドドライバ利用時には必ず最初にこのファンクションを実行しなければならない．各チャネルのプレイバッファを確保して，そのアドレス，サイズを共通制御情報通知域にセットしておくこと.

　本ドライバを使用中は，ドライバ作業域の内容を保証しなければならない.

　このファンクションを実行しただけでは，従来のサウンドBIOSと同じ機能しか使えないので，YM2608の拡張機能を使う場合には，「拡張モードでの初期化(AH=03H)」を実行すること.

PLAY	AH=01H	
演奏の開始		

入　力

AH = 01H
ES:BX = パラメータリストアドレス

出　力

なし

機　能

ES:BXで与えられたパラメータリストに従い，各チャネルの演奏を開始する．

パラメータリストの形式は，以下のとおり．

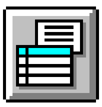
チャネル7～9，リズムは「拡張モードでの初期化(AH=03H)」ファンクションを実行していなければ無効である．

データブロックのサイズは，初期化時に設定したバッファの空きエリアよりも小さくなければならない．バッファの空きエリアよりも大きなデータブロックを与えた場合の動作は保証しない．

各チャネルの演奏終了を判定するには，バッファエンプティ割り込みを利用するか，共通制御情報通知域を参照して有効バイト数を見る必要がある．

パラメータリスト

▼*パラメータリスト=ES:BX(INT 0DAH/01H)*



CLEAR	AH=02H	

演奏の中止とプレイバッファのクリア

入　力

AH = 02H
BL = 制御コード
　　00H ： 演奏を中止して，バッファをクリア
　　01H ： 演奏を中止して，バッファをクリアして，共通制御情報通知域を初期化する

出　力

なし

機　能

　演奏を中止しプレイバッファをクリアする．
　サウンドドライバの利用を終了する場合には，必ずこのファンクションを実行しなければならない．

READ REG	AH=10H	
OPNAレジスタの読み出し1		

入　力

AH = 10H
AL = レジスタ番号(00H～FFH)

出　力

BH = 00H
BL = レジスタ内容

機　能

指定したOPNAレジスタの内容を読み出す.

このファンクションで保証するのは「OPNAレジスタへの書き込み1(AH=11H)」で書き込んだ値についてのみであり,直接書き込んだ場合には保証しない.

このファンクションで読み出せるレジスタはSSG 音源部,リズム部,FM音源部(共通部 ch1～ch3)だけなので,FM音源部(ch6～ch7)のレジスタの内容を読み出すには「OPNAレジスタの読み出し2(AH=20H)」を利用すること.

WRITE REG	AH=11H	

OPNAレジスタへの書き込み1

入　力

AH = 11H
AL = レジスタ番号(00H～FFH)
BL = 設定値

出　力

なし

機　能

指定したOPNAレジスタに値を書き込む.

レジスタの内容を変更するときは，必ずこのファンクションを利用すること.

OPNAの全レジスタの書き込みを許可しているが，タイマー等のドライバの基本動作に関わるレジスタを変更した場合，その後の動作については保証しない.

このファンクションで書き込めるレジスタはSSG音源部，リズム部，FM音源部(共通部 ch1～ch3)だけなので，FM 音源部(ch6～ch7)のレジスタに書き込むには「OPNAレジスタへの書き込み2(AH=21H)」ファンクションを利用すること.

本コマンドは，できるだけ各音源のパラメータの一時的変更の目的に使用すること.

SET TOUCH	AH=12H	
Gate Time/Step Time値の設定		

入　力

AH = 12H
AL = チャネル番号(0～8 ： チャネル1～9に対応)
BL = Gate Time/Step Time値(0～7 ： 1/8～8/8に対応)

出　力

なし

機　能

「音程，音長の設定1(AH=13H)」実行時のGate Time/Step Time値を設定する．

本コマンドは，音調の時間(Step Time)に対し，実際に音を出している時間(Gate Time)を規定する．しかし，時間が12以下の音調に対しては，ドライバ内部で適当に分割するため，必ずしもGate Time/Step Time値で規定した比率とはならない(近い比率にはなる)．また，エンベロープ形状の設定状態によっては，小さいGate Time/Step Time値に設定した際，発音しても聞き取れない場合がある．

サウンドドライバでは音調をStep Timeで指定する．テンポ設定は，このStep Time 48に対する1分間の演奏回数で設定される．1小節を192 Step Timeは4分音符に相当する．

NOTE	AH=13H	

音程，音長の設定1

入　力

AH = 13H
AL = チャネル番号(0～8 ： チャネル1～9に対応)
BH = 音程
　　0～96 ： キー番号
　　128 ： 休符
BL = 音長
　　0 ： 省略(デフォルト値)
　　1～255 ： 音長(Step Time)

出　力

なし

機　能

音程，音長の指定に従って発音する．Gate/Step値は，「Gete Time/Step Time値の設定(AH=12H)」で設定された値をとる．

このファンクションで，リズムパートを発音させることはできない．

リアルタイム機能において，指定チャネル発音中に実行した場合，前の音が切れて新しい音が発音される．本ファンクションは，特別な用法以外ディレイド機能として使用したほうが望ましい．

▼*KEY番号と音程の対応=BH(INT 0DAH/13H)*

SET LENGTH	AH=14H	
デフォルト音長の設定		

入　力

AH = 14H
AL = チャネル番号(0〜8 ： チャネル1〜9に対応)
BL = 音長(1〜255 ： Step Timeで指定)

出　力

なし

機　能

デフォルト音長を設定する.

SET TEMPO	AH=15H	

テンポの設定

入　力

AH = 15H
BL = テンポ(1～255 : 48step/min単位で指定)

出　力

なし

機　能

全チャネル共通にテンポを設定する．単位は1分間に演奏する48 Step Time長の「音程，音長の設定1(AH=13H)」の数である．

あまり大きな値を設定すると，演奏が乱れることがあるので注意すること．

SET PARA BLOCK	AH=16H	

音色パラメータの設定

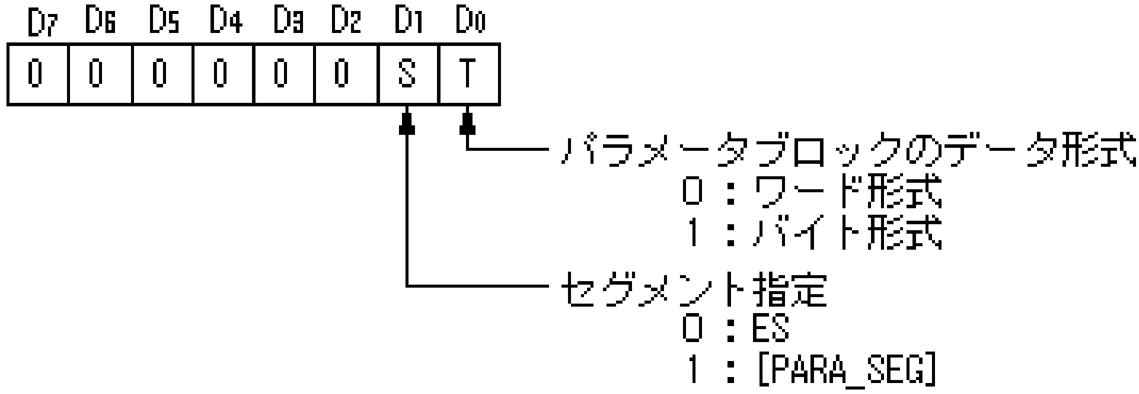
入　力

AH = 16H

AL = チャネル番号(0～8 ： チャネル1～9に対応)

BX = パラメータブロックオフセット

DL = パラメータブロックのデータ形式



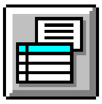
出　力

なし

機　能

　FM音源チャネルに対し音色パラメータを設定する.
　指定チャネルがFM音源でなかった場合は無視される.
　音量は保存されないので，必要があれば再設定すること.
　DLのbit1=1を指定した場合には，セグメントに[PARA_ SEG]を用いるので，あらかじめ共通制御情報通知域に値をセットしておかなければならない.
　音色パラメータ設定直後はLFO効果はONになっている.

▼*パラメータブロック詳細1*



READ PARA	AH=17H	
各パラメータの取得		

入　力

AH = 17H
AL = チャネル番号(0～8 ： チャネル1～9に対応)
BL = パラメータ番号(0～49)

出　力

BX = 設定値

機　能

指定チャネルの音色パラメータの1つを得る．パラメータがバイトの場合，BXの上位バイトには0が入る．

SSG音源のチャネルにFM音源のパラメータを指定した場合に返される値は不定である．

WRITE PARA	AH=18H	
各パラメータの設定		

入　力

AH = 18H
AL = チャネル番号(0〜8 ： チャネル1〜9に対応)
BL = パラメータ番号(0〜49)
DX = 設定値

出　力

なし

機　能

各チャネル番号にパラメータを設定する.
パラメータがバイトの場合，上位バイトは無視される.
SSG音源のチャネルにFM音源のパラメータを設定した場合の動作は保証しない.

ALL STOP	AH=19H	

演奏の一時中断

入　力

AH = 19H

出　力

なし

機　能

　演奏を一時中断する.

　全チャネルのON/OFF状態は保存され，「演奏の再開(AH=1AH)」に備える.このファンクションは，タイマ割り込みを止めるので，再び割り込みを動作させるには「演奏の開始(AH=01H)」または「演奏の再開(AH=1AH)」を実行する.

　演奏を再開する場合，ドライバ作業域を保証しなければならない.

CONT PLAY	AH=1AH	

演奏の再開

入　力

AH = 1AH

出　力

なし

機　能

中断した演奏を再開する.

「演奏の一時中断(AH=19H)」ファンクションが実行されていない場合，このファンクションを呼び出しても，ドライバの動作には何も影響しない.

HOLD STATE	AH=1EH	
KEY ON/OFF状態の維持		

入　力

AH = 1EH
AL = チャネル番号(0～8 ： チャネル1～9に対応)
BL = 維持する長さ
　　0 ： 省略(デフォルト値)
　　1～255 ： Step Time

出　力

なし

機　能

KEY ON/OFF状態を維持する.
「OPNAレジスタへの書き込み1(AH=11H)」で, KEY ON/OFFの状態を変更した後に一定時間その状態を保つ場合, このファンクションを使用する.

MODU ON	AH=1BH	
LFO効果のON		

入　力

AH = 1BH
AL = チャネル番号(0～8 ： チャネル1～9に対応)

出　力

なし

機　能

LFO効果をONにする.
無効なLFOパラメータが設定されていた場合は何もしない.

MODU OFF	AH=1CH	
LFO効果のOFF		

入　力

AH = 1CH

AL = チャネル番号(0～8 ： チャネル1～9に対応)

出　力

なし

機　能

LFO効果をOFFにする.

SET INT COND	AH=1DH	

プレイバッファエンプティ割り込みの条件設定

入　力

AH = 1DH
AL = チャネル番号(0～8，255:チャネル1～9，リズムに対応)
DX = 有効バッファ長
　　bit15　0 : 割り込み禁止
　　　　　 1 : 割り込み許可
　　bit 0～14 : 有効バッファ長
ES:BX = 割り込みルーチンエントリアドレス

出　力

なし

機　能

プレイバッファエンプティ割り込み条件を設定する.

第4章

拡張サウンドファンクション

INITIALIZE2	AH=03H	
拡張モードでの初期化		

入　力

AH ＝ 03H
ES ＝ ドライバ作業域セグメントアドレス
　　プレイバッファ情報10チャネル分
　　　(チャネル7～9，リズムは拡張制御情報通知域内)
　　　BUF_OFS_n ： チャネルnのプレイバッファオフセット
　　　BUF_SEG_n ： チャネルnのプレイバッファセグメント
　　　BUF_LNG_n ： チャネルnのプレイバッファサイズ(バイト)
　　リズムパターンバッファアドレス
　　　RYM_SEG ： リズムパターンバッファセグメント

出　力

BX ＝ ステータス
　　0 ： 正常終了
　　1 ： YM2608が実装されていない

機　能

　サウンドドライバを拡張モードで初期化する.
　サウンドドライバを拡張モードで利用するときには，必ず最初にこのファンクションを実行しなければならない.
　各チャネルのプレイバッファを確保して，そのアドレス，サイズを共通制御情報通知域にセットしておくこと(「ドライバの初期化(AH=00H)」と同様). また，リズムパターンバッファを確保して，そのアドレスもセットしておくこと.
　拡張モードでは，FM 6声+SSG 3声+リズムの10チャネルを使って演奏することができる.

BLOCK PLAY	AH=04H	

ディレイドデータ列の実行

入　力

AH = 04H

AL = 演奏回数

　0 ： エンドレス

　1〜255 ： 演奏回数

ES:BX = パラメータリストアドレス

出　力

なし

機　能

ES:BXで与えられたパラメータリストに従い，各チャネルの演奏をする．パラメータリストの形式は次のとおり．

▼パラメータリスト=ES:BX(INT 0DAH/04H)

このファンクションは「演奏の開始(AH=01H)」と違い，データをプレイバッファに転送せずに与えられたデータをそのまま用いて演奏する．したがって，演奏中はデータ領域を保証しなければならない．

このファンクションで演奏中に「演奏の開始(AH=01H)」を実行してはならない．

ドライバは，FM音源部のデフォルトパラメータを所持していないので，必要があればパラメータをセットしてからこのファンクションを実行すること．

このファンクションは，プレイバッファ情報(BUF_OFS_n，BUF_SEG_n，BUF_LNG_n)を破壊するので，このファンクション実行後に「演奏の開始(AH=01H)」を利用する場合は，プレイバッファ情報を再びセットしなければならない．(「第5章 ディレイド機能フォーマット」参照)

READ REG2	AH=20H	
OPNAレジスタの読み出し2		

入　力

AH = 20H
AL = レジスタ番号(30H～B6H)

出　力

BH = 0
BL = レジスタ内容

機　能

指定したOPNAレジスタの内容を読み出す.

このファンクションで保証するのは「OPNAレジスタへの書き込み2(AH=21H)」で書き込んだ値についてのみであり，直接書き込んだ場合には保証しない.

このファンクションで読み出せるレジスタは，FM音源部(ch7～ch9)のレジスタだけなので，SSG音源部，リズム部，FM音源部(共通部 ch1～ch3)のレジスタの内容を読み出すには「OPNAレジスタの読み出し1(AH=10H)」を利用すること.

互換モードではこのファンクションは利用できない.

WRITE REG2	AH=21H	
OPNAレジスタへの書き込み2		

入　力

AH = 21H
AL = レジスタ番号(30H～B6H)
BL = 設定値

出　力

なし

機　能

指定したOPNAレジスタに値を書き込む.

レジスタの内容を変更するときは，必ずこのファンクションを利用すること.

このファンクションで書き込めるレジスタはFM音源部(ch7～ch9)のレジスタだけなので，SSG 音源部，リズム部，FM音源部(共通部 ch1～ch3)のレジスタに書き込むには「OPNAレジスタの読み出し1(AH=10H)」ファンクションを利用すること.

互換モードではこのファンクションは利用できない.

TIE NOTE	AH=22H	

音程，音長の設定2

入　力

AH = 22H
AL = チャネル番号(0～8 ： チャネル1～9に対応)
BH = 音程
　0～96 ： キー番号
　128 　： 休符
BL = 音長
　0 ： 省略(デフォルト値)
　1～255 ： 音長(Step time)

出　力

なし

機　能

音程，音長の指定に従って発音する．Gate/Step値は，「Gate Time/Step Time値の設定(AH=12H)」で設定された値を無視して必ず8/8に設定される．

次回の「音程，音長の設定1(AH=13H)」，「音程，音長の設定2(AH=22H)」で同じキー番号を指定した場合には，発音動作を行わずに音長だけが設定される．

このファンクションで，リズムパートを発音させることはできない．

GET PARA BLOCK	AH=23H	
音色パラメータの読み出し		

入　力

AH = 23H
AL = チャネル番号(0～8 ： チャネル1～9に対応)
DL = パラメータブロックのデータ形式
　0 ： ワード
　1 ： バイト
ES:BX = 転送先パラメータブロックアドレス

出　力

[ES:BX]パラメータブロック

機　能

FM音源チャネルに対して設定された音色パラメータを，ES:BXで指定されたアドレスに読み出す.

OUT CONTROL	AH=24H	

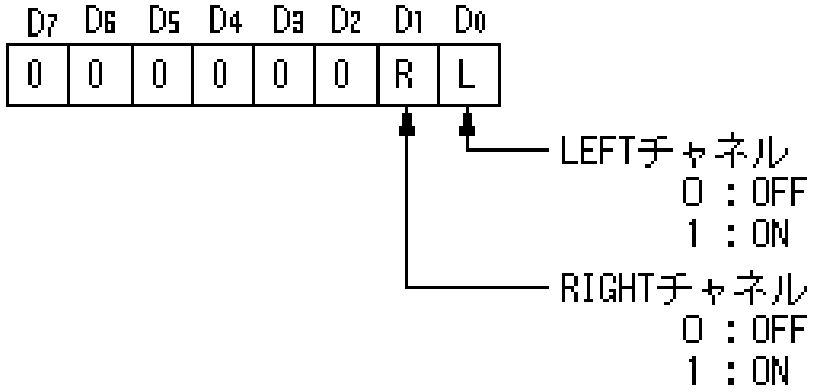
出力チャネルの設定

入　力

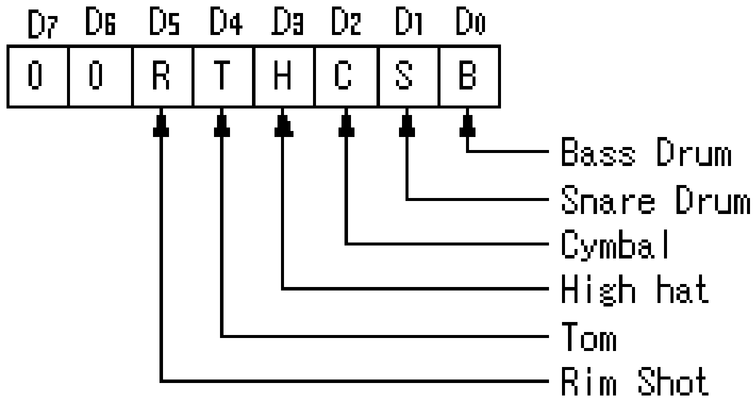
AH = 24H

AL = チャネル番号(0～8，255 ： チャネル1～9，リズムに対応)

BL = 設定値



BH = パートデータ(AL=255の場合のみ)



出　力

なし

機　能

指定チャネルの出力チャネル(RIGHT/LEFT)を制御する.

KEY ON	AH=25H	
発音		

入　力

AH = 25H
AL = チャネル番号(0～8 ： チャネル1～9に対応)
BL = 音程
　　0～96 ： キー番号

出　力

なし

機　能

音程の指定に従って発音する.

KEY OFF	AH=26H	
消音		

入　力

AH = 26H
AL = チャネル番号(0～8 ： チャネル1～9に対応)

出　力

なし

機　能

「発音(AH=25H)」で発音した音を止める.

第5章

ディレイド機能フォーマット

5.1　FMおよびSSGチャネル

●音程，音長の設定1

第1バイト ＝ 音程
　　0～60H ： キー番号
　　80H ： 休符
第2バイト ＝ 音長
　　00H ： 省略(既定値)
　　1～FFH ： 音長

●OPNAレジスタへの書き込み1

第1バイト ＝ 81H
第2バイト ＝ レジスタ番号
第3バイト ＝ 既定値

●Gate Time/Step Time値の設定

第1バイト ＝ 82H
第2バイト ＝ Gate Time/Step Time値(0～7)

●デフォルト音長の設定

第1バイト = 83H
第2バイト = 音長(1～FFH)

●テンポの設定

第1バイト = 84H
第2バイト = テンポ(1～FFH)

●音色パラメータの設定

第1バイト = 85H
第2バイト = パラメータブロックの形式
00H ： ワード
01H ： バイト
第3,4バイト = パラメータブロック開始オフセット
第5,6バイト = パラメータブロック開始セグメント
(パラメータブロックがワード形式の場合のみ)

●各パラメータの設定

第1バイト = 86H
第2バイト = パラメータ番号(0～49H)
第3バイト = 設定値
第4バイト = 設定値

●LFO効果をON

第1バイト = 87H

●LFO効果をOFF

第1バイト = 88H

●KEY ON/OFFの状態の維持

第1バイト = 89H
第2バイト = 維持する長さ
00H ： 省略(既定値)
1～FFH ： 維持する長さ

●音量の設定

第1バイト = 8AH
第2バイト = レジスタ番号
第3バイト = 設定値

●OPNAレジスタへの書き込み2

第1バイト = 8BH
第2バイト = レジスタ番号
第3バイト = 設定値

●音程，音長の設定2

第1バイト = 8CH
第2バイト = 音程
0～60H ： キー番号
80H ： 休符
第3バイト = 音長
00H ： 省略(既定値)
1～FFH ： 音長さ

●出力チャネルの設定

第1バイト = 8DH
第2バイト = 既定値

●発音

第1バイト = 8EH
第2バイト = 音程(0～60H ： キー番号)

●消音

第1バイト = 8FH

5.2　リズムチャネル

●リズムパターンの設定

第1バイト = リズムパターン番号

●リズムボリュームの設定

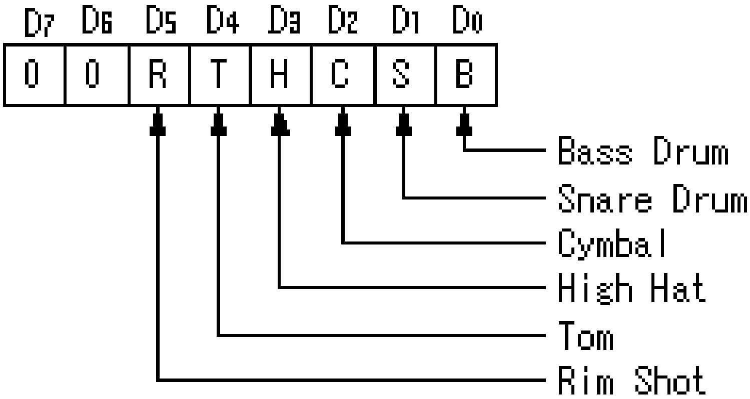
第1バイト = 音量(80H～BEH ： 0～63に対応)

■リズムパターンデータ

リズムパターンデータは基本的に1つのコマンドが4分音符の1/12の長さに相当する.

●発音

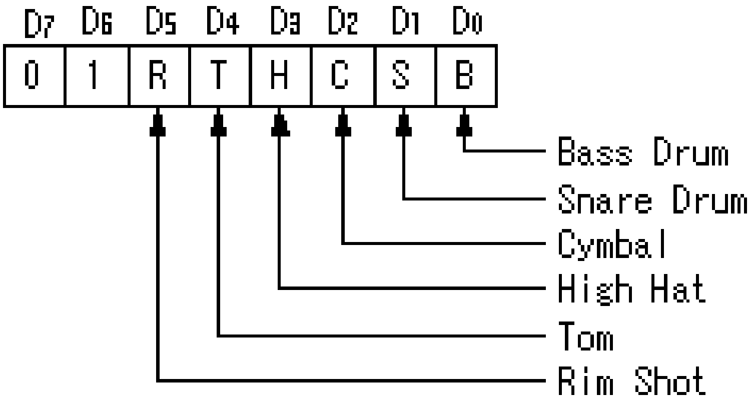
第1バイト = パート指定



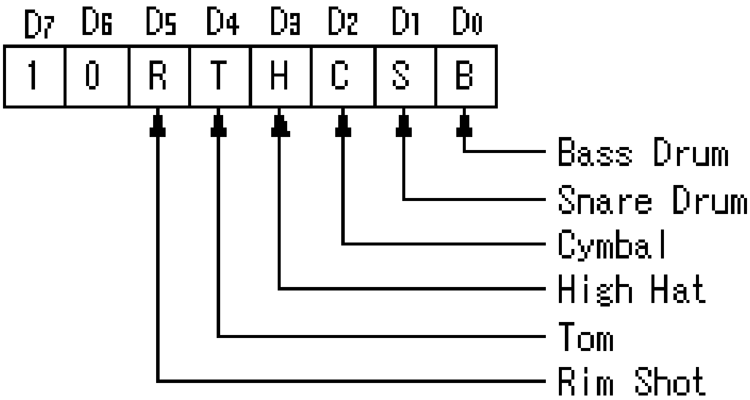
bitが1になっているパートをKEY ON する.

●発音と消音

第1バイト = パート指定(発音)



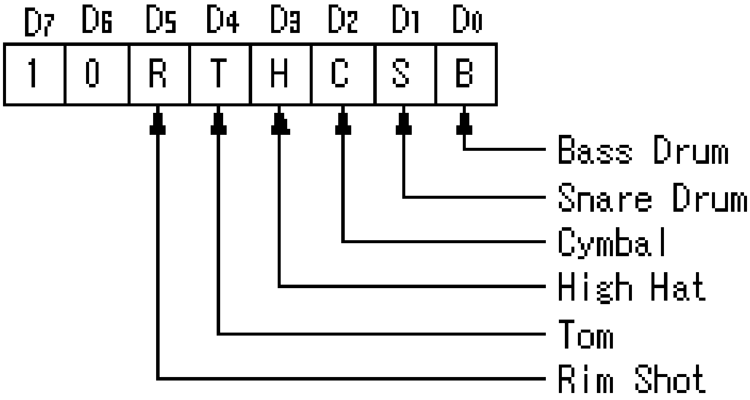
第2バイト = パート指定(消音)



bitが1になっているパートをKEY ONする.

●消音

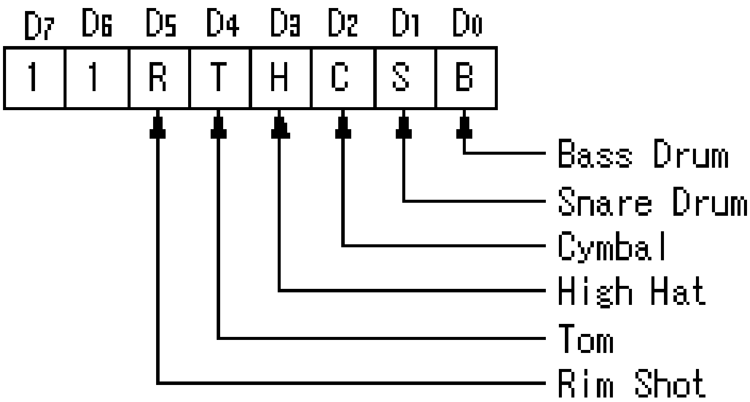
第1バイト ＝ パート指定



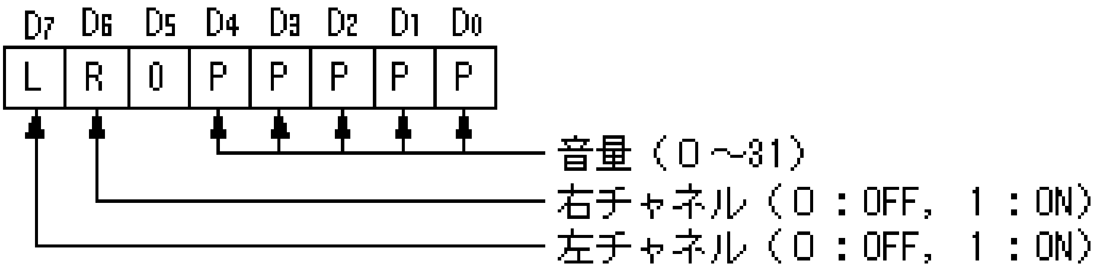
bitが1になっているパートをDUMPする.

●出力チャンネルの設定

第1バイト ＝ パート指定



第2バイト



第4部
PCMドライバ

第1章

PCM/DSPドライバ サポートファンクション

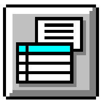
PCM/DSPドライバは，PCMの録音/再生およびDSP(μPD6380)を用いた音場生成を制御するものである(DSPはPC-98GSおよびPC-9801-73にのみ搭載)．

DSPを用いることにより，各種内蔵音源からのサウンドソースに特殊効果を加えて出力させることができる．

このドライバは80286以上のCPUをターゲットとしたものであり，V30モードでの動作は保証されない．

●PCM/DSPドライバファンクション一覧(INT 0D9H)

▼*PCM/DSPドライバファンクション一覧*

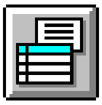


第2章

エラーステータス

　PCM/DSPドライバでは，コマンドの終了状態をメインプログラムで確認するため，Carryフラグにエラーの有無を，またAHにエラー発生時のステータスを返す．以下にこのステータスの内訳を示す．

▼*PCM/DSPドライバエラーステータス一覧*



第3章

コマンドの相互関係

以下にコマンドの相互関係を示す.

例えばPCM/ADPCMの録音/再生(AH=10H)コマンドを実行する場合，その前に「PCM/DSPドライバの初期化(AH=00H)」，FM音源の「ドライバの初期化(INT 0DAH, AH=00H)」と「ファンクションの初期化(AH=01H)」，「パラメータの設定(AH=03H)」，「ミキシングレートの設定(AH=07H)」の4つのコマンドを実行しておく必要がある(ただしデフォルトの設定でよい場合は「PCM/DSPドライバの初期化(AH=00H)」，FM音源の「ドライバの初期化(INT 0DAH, AH=00H)」のみ).

▼*コマンドの相互関係*

第4章

ファンクション詳細

$INITDRV	AH=00H	
PCM/DSPドライバの初期化		

入　力

AH = 00H

出　力

Carry = 終了ステータス
　　0 ： 正常終了
　　1 ： エラー
AH = エラーステータス
　　0FH ： タイムアウトエラー

機　能

PCM/DSPドライバ全体のイニシャライズを行う．すなわち，ドライバ内部で後述の各コマンドルーチンを呼び出し，各々について以下のような設定を行う．

①ミュート状態の設定(AH=09H)
音声出力をミュートON状態にする

②ファンクションの初期化(AH=01H)(GS/PC-9801-73のみ)
DSPでスルー($Thru)を実行する

③パラメータの設定(AH=03H)(GS/PC-9801-73のみ)
スルーのフロントL～リアR出力を最大値に設定する

④入力ボリューム値の設定(AH=05H)
VOL1～VOL5をボリューム値0AHに設定する

⑤ミキシングレートの設定(AH=07H)
PCM/DSPの入力ミキシングレートを50%に設定する

⑥サンプリング周波数の設定(AH=0BH)
サンプリング周波数を44.1kHzに設定する

⑦PCMモードの設定(AH=0DH)
出力モード0，ステレオ，16ビットデータ長に設定する

⑧ミュート状態の設定(AH=09H)
音声出力をミュートOFF状態にする

注　意

FM/SSGドライバとPCMドライバは同一の領域を利用している．そのため，他のPCM/DSPファンクションを利用する前には本ファンクションを実行し，FM/SSGドライバのファンクション「ドライバの初期化(INT 0DAH，AH=00H)」もしくは「拡張モードでの初期化(INT 0DAH，AH=03H)」のいずれかを実行して，両方を初期化しておく必要がある．

$INITFUNC	AH=01H	GS73

ファンクションの初期化

入　力

AH = 01H
AL = DSPファンクション
　　「第5章 DSPファンクション」参照

出　力

Carry = 終了ステータス
　　0 ： 正常終了
　　1 ： エラー
AH = エラーステータス(02H, 07H, 0FH)
　　正常終了時は01H

機　能

ファンクションの初期化を実行する.

このコマンドを実行すると, 元のファンクションの内部パラメータ(「パラメータの設定(AH=03H)」で設定)が新しいファンクションのデフォルト値に置き替わる.

ただし, その他のコマンドの設定には影響しない.

注　意

　コマンドを実行しても音が発生しない場合は，「入力ボリューム値の設定(AH=05H)」，「ミュート状態の設定(AH=09H)」コマンドを用いてボリュームとミュートの状態を変更する必要がある.

　「PCMモードの設定(AH=0DH)」の入出力モードが0に設定されているとき, DSPファンクション「ADPCM録音/再生(12H)」のロードを指定するとエラーとなる.

　録音/再生中にこのコマンドを実行するとエラーとなる.

$ASKFUNC	AH=02H	GS73

初期化されているファンクションの取得

入　力

AH = 02H

出　力

Carry = 終了ステータス
　　　　0 ： 正常終了
　　　　1 ： エラー
AH = エラーステータス(02H, 07H, 0FH)
　　正常終了時は02H

機　能

現在初期化されているファンクション番号を取得する.
ただし, その他のコマンドの設定には影響しない.

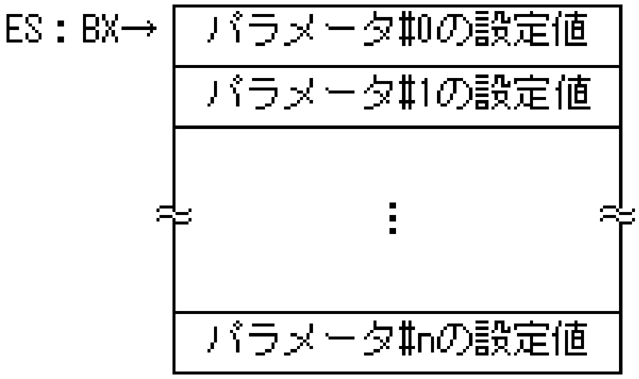
$SETPARAM	AH=03H	GS73

パラメータの設定

入　力

AH = 03H

ES:BX = パラメータブロックのアドレス



出　力

Carry = 終了ステータス

　　0 ： 正常終了

　　1 ： エラー

AH = エラーステータス(02H, 07H, 0FH)

　　正常終了時は03H

機　能

初期化されているファンクションのパラメータ設定を行う．コマンド実行前にパラメータブロックエリアにパラメータを格納しておく必要がある．パラメータブロックの詳細については「第5章 DSPファンクション」を参照のこと．

注　意

録音/再生中にこのコマンドを実行するとエラーとなる．

$GETPARAM	AH=04H	GS73

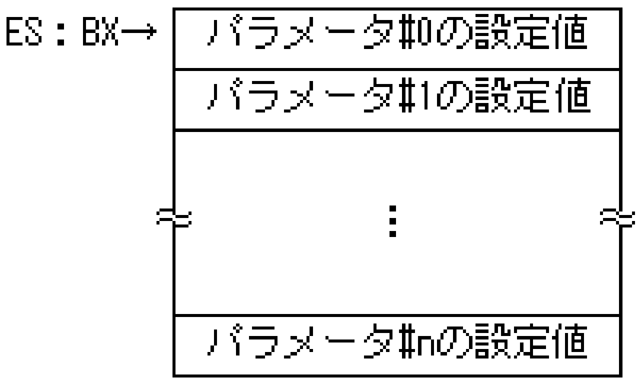
パラメータの取得

入　力

AH = 04H

出　力

Carry = 終了ステータス
　　0 ： 正常終了
　　1 ： エラー
AH = エラーステータス(02H, 07H, 0FH)
　　正常終了時は04H
CX = パラメータの総数(n)
ES:BX = パラメータブロックのアドレス



機　能

初期化されているファンクションのパラメータ取得を行う.

パラメータブロックの詳細については「第5章 DSPファンクション」を参照のこと.

注　意

録音/再生中にこのコマンドを実行するとエラーとなる.

$SETVOL	AH=05H	

入力ボリューム値の設定

入　力

AH = 05H

AL = ボリューム設定

D7	D6	D5	D4	D3	D2	D1	D0
0	b6	b5	b4	b3	b2	b1	b0

b3～b0：ボリューム値
0～FH（0：最小）

b6～b4：ボリューム選択
0：VOL1（YM2608 PCM 処理なし）
1：VOL2（YM2608 PCM 処理あり）
2：VOL3（LINE入力 PCM 処理なし）
3：VOL4（LINE入力 PCM 処理なし）
4：VOL5（CD-DA 再生出力）（PC-98GSのみ）
→「第1部 マルチメディア概要
3.1 ハードウェア
■PCM/DSP 機能」ブロック図参照

出　力

Carry = 終了ステータス
0 ： 正常終了
1 ： エラー

AH = エラーステータス(02H)
正常終了時は05H

機　能

ハードウェア構成図の入力ボリューム(VOL1～VOL5)のレベル調整を行う．ボリューム値は0Hが最小レベルを，FHが最大レベルを表す．

なお，AL=5(または4)XH～FXHは指定不可．

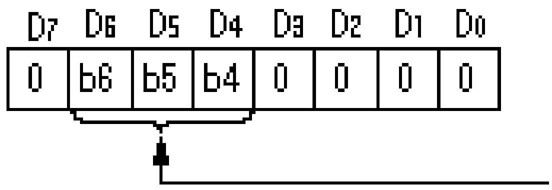
$GETVOL	AH=06H	

入力ボリューム値の取得

入　力

AH = 06H

AL = ボリューム選択

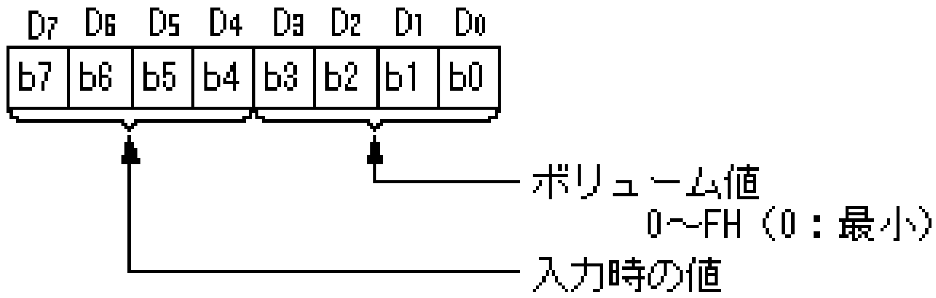


ボリューム選択
0：VOL1（YM2608 PCM 処理なし）
1：VOL2（YM2608 PCM 処理あり）
2：VOL3（LINE入力 PCM 処理なし）
3：VOL4（LINE入力 PCM 処理あり）
4：VOL5（CD-DA 再生出力）（PC-98GS）

→「第1部 マルチメディア概要
3.1 ハードウェア
■PCM/DSP 機能」ブロック図参照

出　力

Carry = 終了ステータス
　　0 ： 正常終了
　　1 ： エラー
AH = エラーステータス(02H)
　　正常終了値は06H
AL = ボリューム設定



機　能

設定された入力ボリューム(VOL1～VOL5)値を取得する．ボリューム値は0Hが最小レベルを，FHが最大レベルを表す．

なお，AL=5(または4)XH～FXHは指定不可．

$SETMIX	AH=07H	

ミキシングレートの設定

入　力

AH = 07H

AL = ミキシングレート

00H : IN1 …100%,　IN2 …　0%
01H : 〃 … 90%,　〃 … 10%
02H : 〃 … 80%,　〃 … 20%
03H : 〃 … 70%,　〃 … 30%
04H : 〃 … 60%,　〃 … 40%
05H : 〃 … 50%,　〃 … 50%
06H : 〃 … 40%,　〃 … 60%
07H : 〃 … 30%,　〃 … 70%
08H : 〃 … 20%,　〃 … 80%
09H : 〃 … 10%,　〃 … 90%
0AH : 〃 …　0%,　〃 …100%
IN1 : CD-DA(PC-98GSのみ)/PCM出力(入出力モード2)
IN2 : FM+LINE-IN/PCM出力(入出力モード1)
(「第1部 マルチメディア概要 3.1 ハードウェア ■PCM/DSP機能」ブロック図参照)

出　力

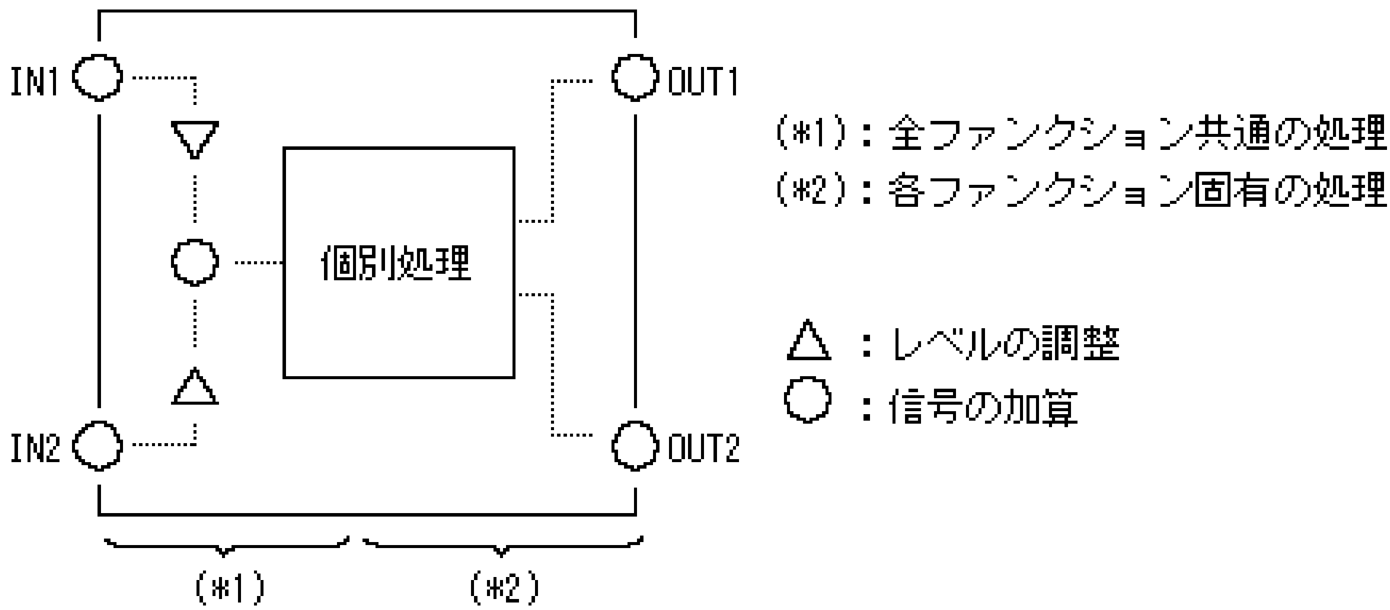
Carry ＝ 終了ステータス
　　0 ： 正常終了
　　1 ： エラー
AH ＝ エラーステータス(02H, 07H)
　　正常終了時は07H

機　能

PCM/DSPのIN1/IN2端子(CD-DA/FM+LINE-IN)からの入力値のミキシングレートを指定する．当ドライバに含まれるすべてのファンクションがこのミキシングに対応している．

なお，IN1/IN2に割り当てられる信号は入出力モードの選択状態によって異なる．詳細は「PCMモードの設定(AH=0DH)」を参照.

PC-98GS以外ではIN1が存在しないため，IN2の入力ボリューム調整として機能する．



注　意

- 44.1kHz以外のサンプリングレートを選択した場合には，CD-DAの入力信号がミュートされる．この操作は，IN1のミキシングレートを0%に落とすことで行われる（IN2のミキシングレートは本来の状態を保つ）．(GS/PC-9801-73のみ)
- 録音/再生中にこのコマンドを実行するとエラーとなる．

$GETMIX	AH=08H	

ミキシングレートの取得

入　力

AH = 08H

出　力

Carry = 終了ステータス
　　0 ： 正常終了
　　1 ： エラー
AH = エラーステータス(02H, 07H)
　　正常終了時は08H
AL = ミキシングレート

00H ：	IN1	…	100%,	IN2	…	0%
01H ：	〃	…	90%,	〃	…	10%
02H ：	〃	…	80%,	〃	…	20%
03H ：	〃	…	70%,	〃	…	30%
04H ：	〃	…	60%,	〃	…	40%
05H ：	〃	…	50%,	〃	…	50%
06H ：	〃	…	40%,	〃	…	60%
07H ：	〃	…	30%,	〃	…	70%
08H ：	〃	…	20%,	〃	…	80%
09H ：	〃	…	10%,	〃	…	90%
0AH ：	〃	…	0%,	〃	…	100%

IN1 ： CD-DA(PC-98GSのみ)/PCM出力(入出力モード2)
IN2 ： FM+LINE-IN/PCM出力(入出力モード1)
(「第1部 マルチメディア概要 3.1 ハードウェア ■PCM/DSP機能」ブロック図参照)

機　能

設定されたPCM/DSPのIN1/IN2端子(CD-DA/FM+LINE-IN)のミキシングレートの値を取得する．当ドライバに含まれるすべてのファンクションがこのミキシングに対応している．

PC-98GS以外ではIN1が存在しないため，IN2の入力ボリューム調整として機能する．

なお，IN1/IN2に割り当てられる信号は入出力モードの選択状態によって異なる．詳細は「PCMモードの設定(AH=0DH)」コマンドを参照.

注　意

・44.1kHz以外のサンプリングレートを選択した場合には，CD-DAの入力信号がミュートされる．この操作は，IN1のミキシングレートを0%に落とすことで行われる(IN2のミキシングレートは本来の状態を保つ)．(GS/PC-9801-73のみ)

・録音/再生中にこのコマンドを実行するとエラーとなる.

$SETMUTE	AH=09H	

ミュート状態の設定

入　力

AH = 09H

AL = ミュート状態

　00H ： ミュートOFF

　01H ： ミュートON

出　力

Carry = 終了ステータス

　0 ： 正常終了

　1 ： エラー

AH = エラーステータス(02H, 0FH)

　正常終了時は09H

機　能

PCM/DSPの出力ミュートのON/OFFを行う．このコマンドにより，フロント・リアスピーカーのミュート状態が同時に切り替わる．

$GETMUTE	AH=0AH	
ミュート状態の取得		

入　力

AH = 0AH

出　力

Carry = 終了ステータス
　　0 ： 正常終了
　　1 ： エラー
AH = エラーステータス(02H, 0FH)
　　正常終了時は0AH
AL = ミュート状態
　　00H ： ミュートOFF
　　01H ： ミュートON

機　能

現在のミュートON/OFF状態を取得する.

$SETFREQ	AH=0BH	
サンプリング周波数の設定		

入　力

AH = 0BH

AL = サンプリング周波数

00H ： 44.10kHz（CD-DAの場合のみ）

01H ： 33.08kHz

02H ： 22.05kHz

03H ： 16.54kHz

04H ： 11.03kHz

05H ： 8.27kHz

06H ： 5.52kHz

07H ： 4.13kHz

出　力

Carry = 終了ステータス

0 ： 正常終了

1 ： エラー

AH = エラーステータス（02H, 07H）

正常終了時は0BH

機　能

PCM/DSPのサンプリング周波数を指定する．「PCM/ADPCMの録音/再生(AH=10H)」により録音/再生を行う場合，サンプリング周波数を低くするほど長時間の録音/再生が可能になる．

また，ディレイの関係するファンクションを実行した場合，周波数を低くするほどディレイ時間を長くすることができる．

ただし，周波数を下げるとその分だけ音質は低下する．

注　意

・サウンドソースとしてCD-DAを用いる場合は，必ず44.1kHzの周波数を指定する必要がある．それ以外の周波数を用いた場合，CD-DAからDSPに供給される信号がミュートされる(詳細は「ミキシングレートの設定(AH=07H)」コマンド参照)．

ただし，FM+LINE-IN信号やI/O出力信号はミュートの影響を受けない．また，CD-DAから直接(VOL5を介して)LINE-OUTに供給される．信号もミュートの影響を受けない．

・録音/再生中にこのコマンドを実行するとエラーになる．

・PC-98GSではDSPを利用してエフェクトをかける場合には，サンプリング周波数は44.10kHzまで利用できる．これはCD-DAの入力に対してエフェクトをかけるときに使う．

PC-98GSではサンプリング時には22.05kHzが上限である．これ以上のサンプリング周波数を設定した場合には，実行時に「PCM/ADPCMの録音/再生(AH=10H)」を使ったときにエラーが返される．このファンクション自体はエラーを返さないので注意が必要である．

$GETFREQ	AH=0CH	

サンプリング周波数の取得

入　力

AH = 0CH

出　力

Carry = 終了ステータス
　　0 ： 正常終了
　　1 ： エラー
AH = エラーステータス(02H, 07H)
　　正常終了時は0CH
AL = サンプリング周波数
　　00H ： 44.10kHz(CD-DAの場合のみ)
　　01H ： 33.08kHz
　　02H ： 22.05kHz
　　03H ： 16.54kHz
　　04H ： 11.03kHz
　　05H ： 8.27kHz
　　06H ： 5.52kHz
　　07H ： 4.13kHz

機　能

設定されたPCM/DSPのサンプリング周波数値を取得する.

注　意

録音/再生中にこのコマンドを実行するとエラーになる.

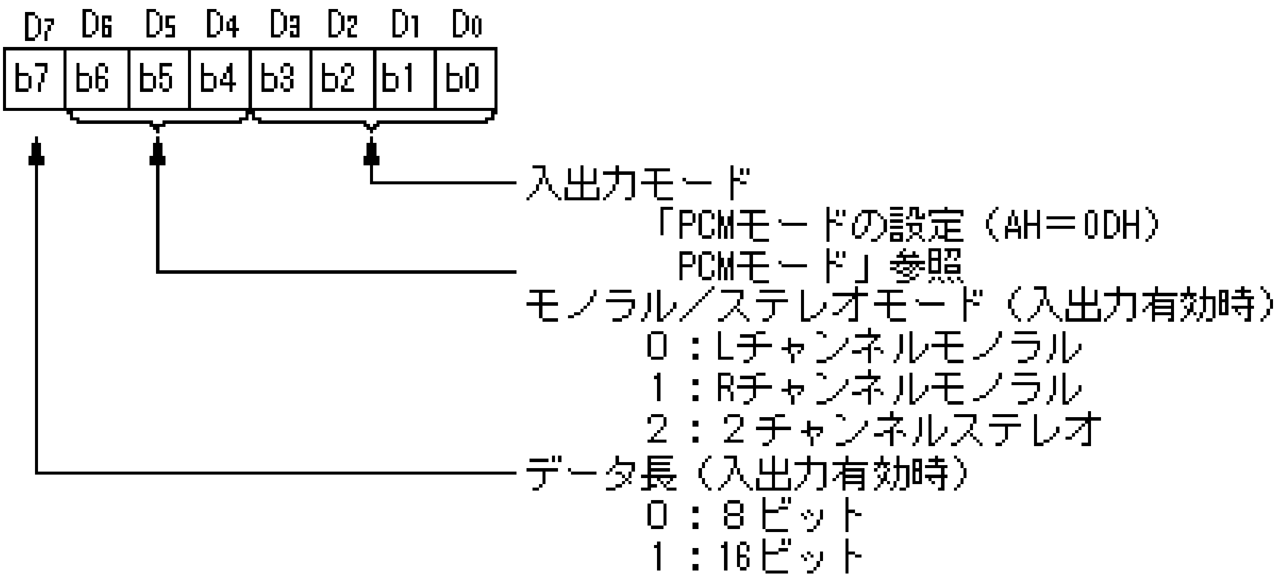
$SETPCM	AH=0DH	

PCMモードの設定

入　力

AH = 0DH

AL = PCMモード



出　力

Carry = 終了ステータス

0 ： 正常終了

1 ： エラー

AH = エラーステータス(02H, 07H, 0FH)

正常終了時は0DH

機　能

PCM入出力の方式を指定する.
コマンド実行時には, ALにPCMモードを指定する.

PCMモード

●入出力モード

PCM入出力を行う場合のPCM/DSP端子(ハードウェア構成図のIN1～OUT2)を選択する. (「第1部 マルチメディア概要 3.1 ハードウェア ■PCM/DSP機能」ブロック図参照)
モード1～8(入出力有効)を選択した場合, サウンドソースとしてはCD-DAとFM+LINE-INのどちらか一方のみが有効となる(この選択はモードにより行うことができる. CD-DAはPC-98GSの場合のみ有効).
また, PCM入力時には, 各々のサウンドソースについて, 以下の(A)～(C)のいずれかを選択することができる.

(A)PCM/DSPへの入力信号
(B)PCM/DSPのフロント出力信号
(C)DSPのリア出力信号 PC-98GS/PC-9801-73の場合のみ有効

これらをまとめると下表のようになる.

モード	CD-DA (*1)	FM+LINE-IN	PCM 出力	PCM 入力
0	IN1(*1)	IN2	×	×
1	IN1(*1)	×	IN2	×
2	× (*1)	IN2	IN1	×
3(*1)	(*1)			IN1
4(*1)	IN1(*1)	×	×	OUT1
5(*2)	(*2)	(*2)	(*2)	OUT2(*2)
6	(*1)			IN2
7	× (*1)	IN2	×	OUT1
8(*2)	(*2)	(*2)	(*2)	OUT2(*2)

(*1)PC-98GS のみ有効

(*2)PC-98GS/PC-9801-73 のみ有効

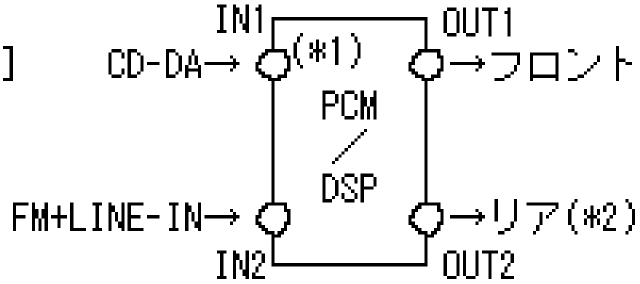
●PCM/DSP入出力モード

入出力モード0～8におけるDSP入出力の様子を以下に示す.

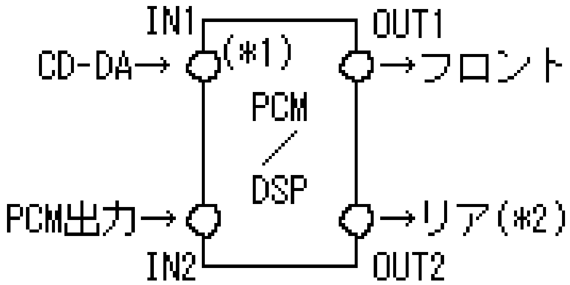
＊1はPC-98GSで有効

＊2はPC-98GS/PC-9801-73で有効

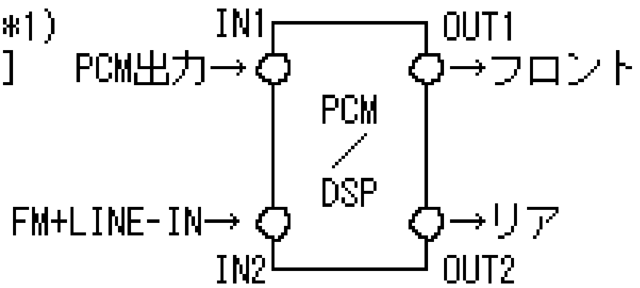
[モード0]



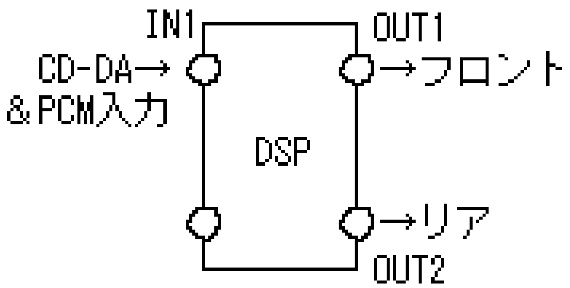
[モード1]



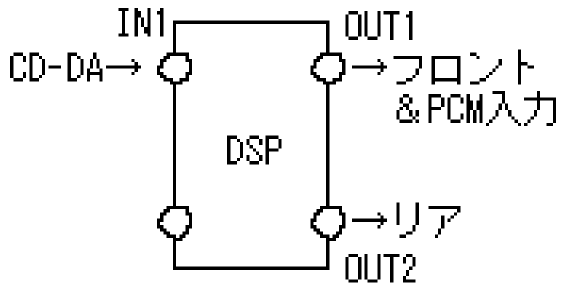
(*1)
[モード2]



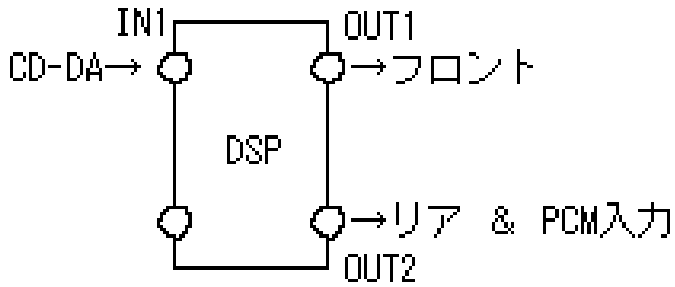
(*2)
[モード3]



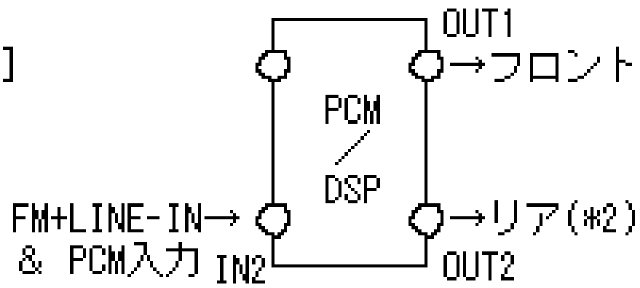
(*2)
[モード4]



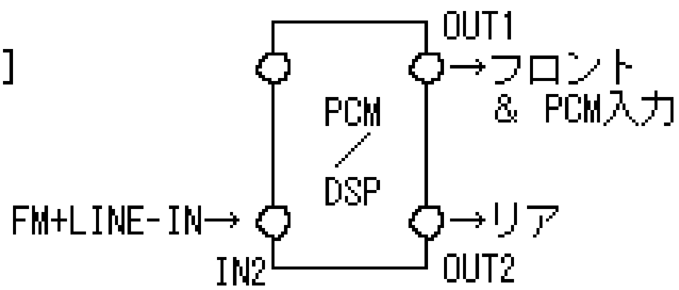
(*2)
[モード5]



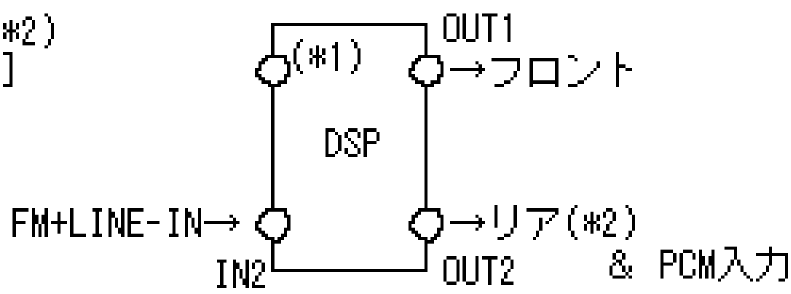
[モード6]



[モード7]



(*2)
[モード8]



●モノラル/ステレオモード

PCM入出力をL/R両チャネルについて行うか，片チャネルのみについて行うかを指定する．片チャネルモードを選択した場合，入力時にはL/Rのいずれか一方のみの信号が取得され，また出力時にはL/Rスピーカのどちらかにのみ信号が出力される．

●データ長

PCM入出力時のデータ長を指定する．

16ビットデータ長を指定した場合，データは上位バイト/下位バイトの順となる．

$GETPCM	AH=0EH	

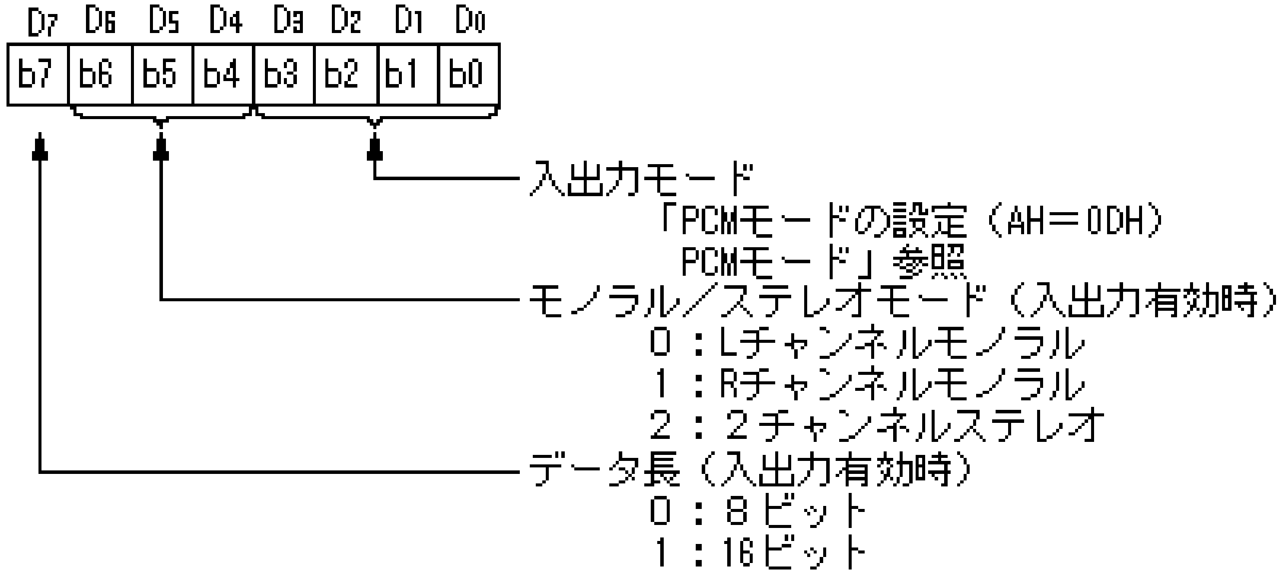
PCMモードの取得

入　力

AH = 0EH

出　力

Carry = 終了ステータス
　　　　0 ： 正常終了
　　　　1 ： エラー
AH = エラーステータス(02H, 07H, 0FH)
　　　正常終了時は0EH
AL = PCMモード(「PCMモードの設定(AH=0DH)」参照)



機　能

設定されたPCM入出力の方式を取得する.

$PCMTRN	AH=10H	

PCM/ADPCMの録音/再生

入　力

AH = 10H

ES:BX = PCB (プレイ・コントロールブロック)アドレス

オフセット	データ長	内　容
+00	1バイト	PCMモード
+01	1バイト	サンプリング周波数
+02	1バイト	使用メモリ(メイン/EMS)
+03	1バイト (*1)	終了ステータス (*1)
+04	1ワード (*1)	EMMハンドル(EMS使用時のみ) (*1)
+06	2ワード	録音/再生のバッファアドレス
+0A	2ワード	録音/再生のバッファサイズ
+0E	2ワード (*1)	録音/再生点のアドレス (*1)
+12	2ワード (*1)	録音/再生済のデータサイズ(バイト) (*1)

(*1) ドライバが値を設定

出　力

Carry = 終了ステータス

0 : 正常終了

1 : エラー

AH = エラーステータス(02H*, 03H, 04H, 05H, 06H, 0FH)

* PC-98GSでサンプリング周波数が33.08kHz以上の場合を含む

正常終了時は10H

機　能

ファンクションの指定に応じて，PCM録音/PCM再生を行う．

PCM録音は，1サンプリング周期ごとにL/Rの各々のデータを指定されたデータバッファ上に格納する．またPCM再生は，データバッファ上の録音データを元に，元の音声を再生する．

コマンドは録音/再生の終了を待たずにメインプログラムに戻るため，その終了判定は，コントロールブロック終了ステータスをモニタするか，「録音/再生ステータスの取得(AH=12H)」コマンドを実行することにより行う必要がある．

録音/再生中は以下のコマンドが受け付けられないため，あらかじめこれらの設定を終えてから「PCM/ADPCMの録音/再生(AH=10H)」コマンドを実行する必要がある．

ファンクションの初期化(AH=01H)(GS/PC-9801-73のみ)
パラメータの設定(AH=03H)(GS/PC-9801-73のみ)
ミキシングレートの設定(AH=07H)
PCMモードの設定(AH=0DH)

コマンド実行時には，ES:BX にプレイ・コントロールブロック(PCB)のアドレスを指定する．

なお，複数バッファにまたがる連続録音/再生(ストリーミング)を実現するため，ドライバはPCBのキューイング機能をサポートしている．すなわち，コマンドをいくつかまとめて実行すると，そのPCBがドライバ内部のキューに登録され(2番目以降はペンディング要求となる)，順次，音を途切らせることなく処理していく．詳細については後述する．

注　意

EMSメモリを使用したストリーミングは不可．

PCB

■PCBの構成

①PCMモード(1バイト)

「PCMモードの設定(AH=0DH)」のPCMモードと基本的に同一である．ただし，以下の例外事項が加わる．

・入出力モード

なお，モード0を指定するとエラーになる．

・モノラル/ステレオモード

モノラル再生を指定した場合には，無条件でモノラル → ステレオ再生(「PCMモードの設定(AH=0DH)」の「入出力処理」を参照)を行う．すなわち，同一のモノラル信号をL/R双方のスピーカーに出力する．

・データ長

ADPCM録音/再生用のファンクションを選択した場合には，無条件で4ビットのデータ長を採用する．このときのバッファ上のデータフォーマットについては後述する．(GS/PC-9801-73のみ)

なお，このフィールドにFFHを指定すると，コマンド実行時に現在の「PCMモードの設定(AH=0DH)」の設定値がコピーされる．

②サンプリング周波数(1バイト)

このフィールドにFFHを指定すると，コマンド実行時に現在の「サンプリング周波数の設定(AH=0BH)」の設定値がコピーされる．なお，CD-DAからDSPへの信号は44.1kHz以外ではミュートされるため(「ミキシングレートの設定(AH=07H)」参照)，CD-DAの録音やCD-DAとPCM出力のミキシング再生を行うことはできない．(GS/PC-9801-73のみ)

③使用メモリ(1バイト)

バッファとして用いるメモリの種類を指定する.

00H ： メインメモリ
01H ： EMSメモリ

④終了ステータス(1バイト)

録音/再生の終了ステータスを返す. また, 録音/再生中はFFHを返す. このフィールドはドライバ側が設定するため, コマンド発行時の値は何であってもよい.

00H ： 正常終了
01H ： 強制終了
02H ： EMSエラー
03H ： タイムアウトエラー
04H ： 転送エラー
FFH ： 録音/再生中

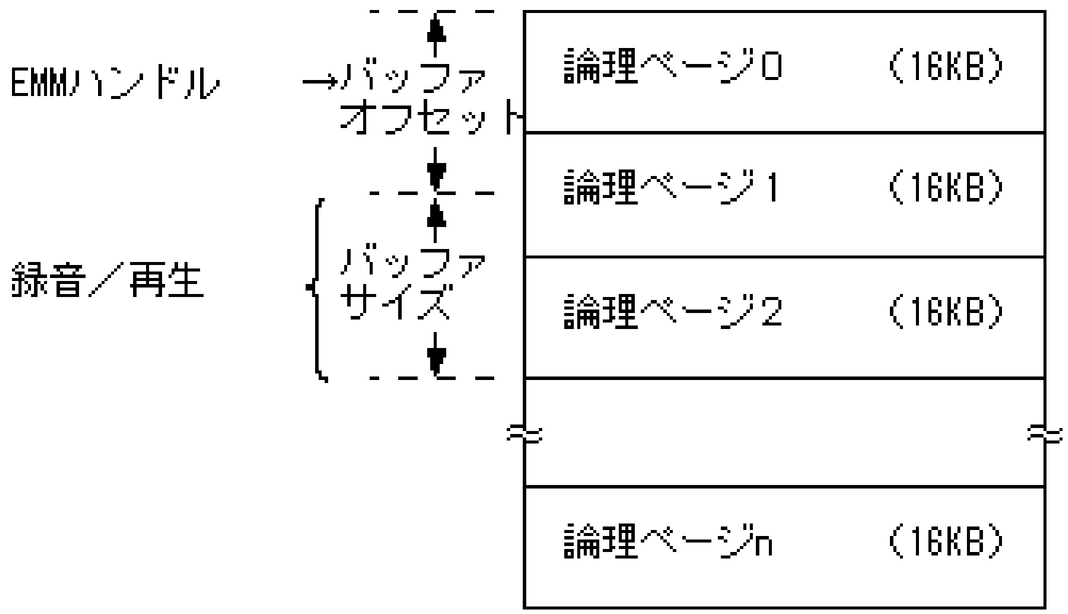
なお, このフィールドは現行のPCBに対するステータスを返すだけであり, ペンディング要求まで含めたステータスを返すわけではない. もし, ペンディング要求が存在する場合には, 「録音/再生ステータスの取得(AH=12H)」コマンドを用いてステータスを取得する必要がある.

⑤EMMハンドル(1ワード)

③でEMSメモリを選択した場合にのみ指定する. このハンドルは, コマンド実行前にEMMを用いて取得しておく必要がある.

⑥録音/再生のバッファアドレス(2ワード)

メインメモリ選択時には，録音/再生用データバッファの開始アドレスを指定する．またEMSメモリ選択時には，データバッファの開始オフセット(論理ページ0の先頭からのオフセット)を指定する．EMS選択時のバッファのイメージを以下に示す．



なお，16ビットのPCM録音/再生を選択した場合，バッファはワード境界から指定する必要がある．

⑦録音/再生のバッファサイズ(2ワード)

⑥のバッファサイズを指定する．ペンディング要求が存在しない場合，このサイズがそのまま録音/再生のデータサイズとなる．サイズには最低8192バイトを確保する必要がある．また，16ビットのPCM録音/再生を選択した場合には，偶数値を指定する必要がある．

⑧録音/再生点のアドレス(2ワード)

現行の録音/再生点のアドレスを返す．このフィールドはドライバ側が設定し，コマンド実行時には⑥の値がコピーされる．また，録音/再生の終了時には⑥+⑦の値を返す．

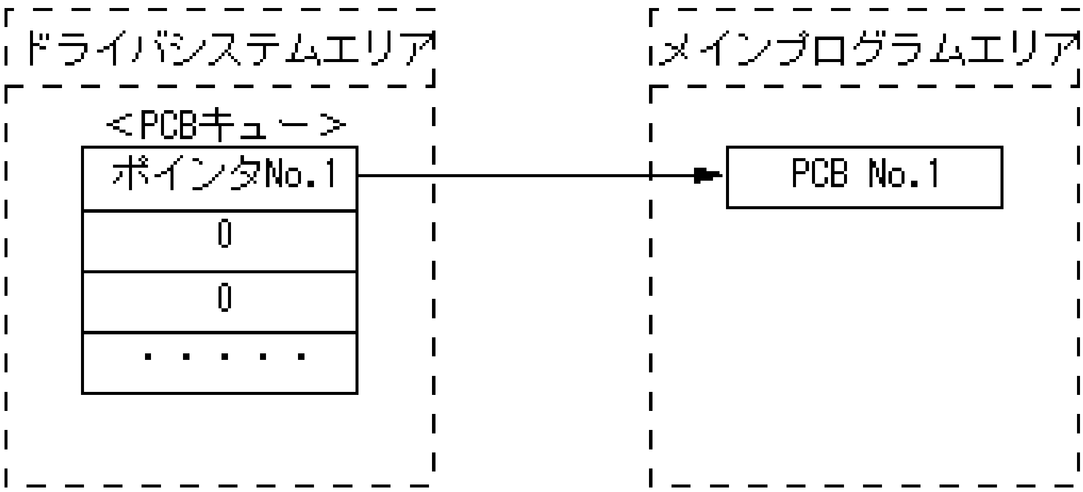
⑨録音/再生済のデータサイズ(2ワード)

録音/再生済のバッファ・データのサイズを返す．このフィールドもドライバ側が設定し，コマンド実行時にはゼロが格納される．また，録音/再生の終了時には⑦の値を返す．

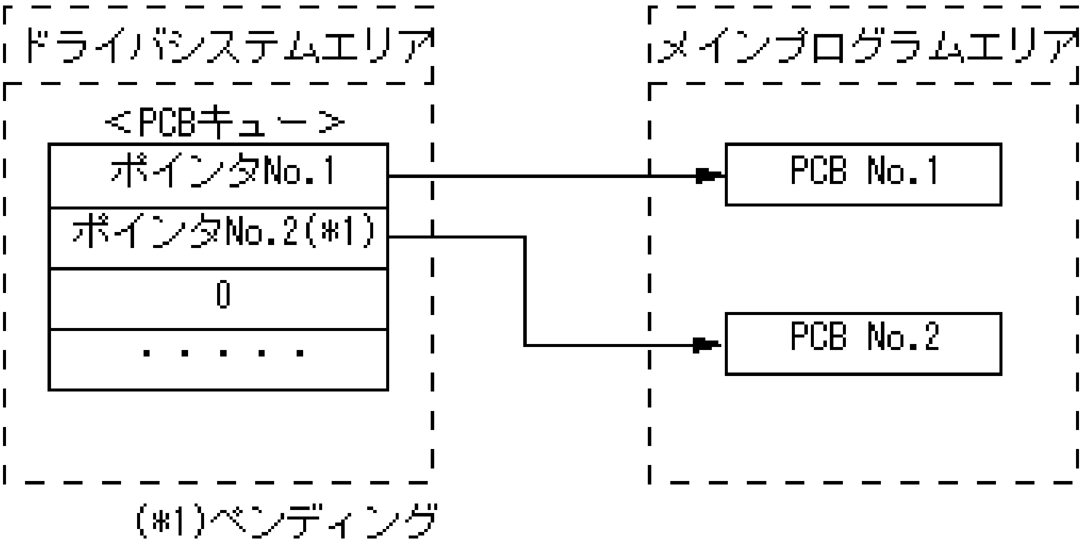
●PCBキュー

ドライバのシステムエリアには，「PCM/ADPCMの録音/再生(AH=10H)」専用のキューが用意されているため，録音/再生中であっても後続の本ファンクションをペンディング要求として登録することができる．
このキューはPCBのポインタから成り，同時に16個までのポインタを格納することができる．このポインタは対応するPCBの録音/再生処理が終了するまで保持され，終了ステータスが設定された時点でキューから削除される．
この様子を以下に示す．

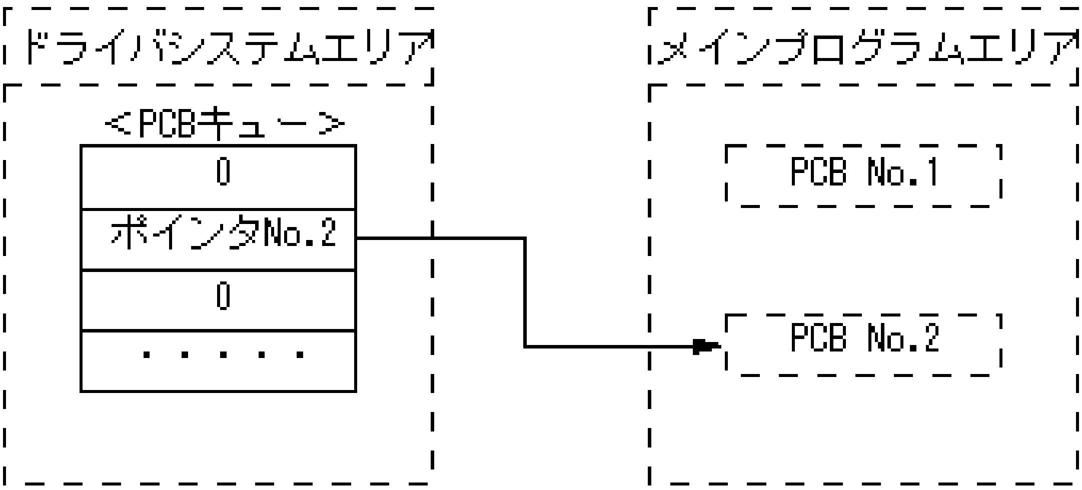
① ファンクションNo.1の発行



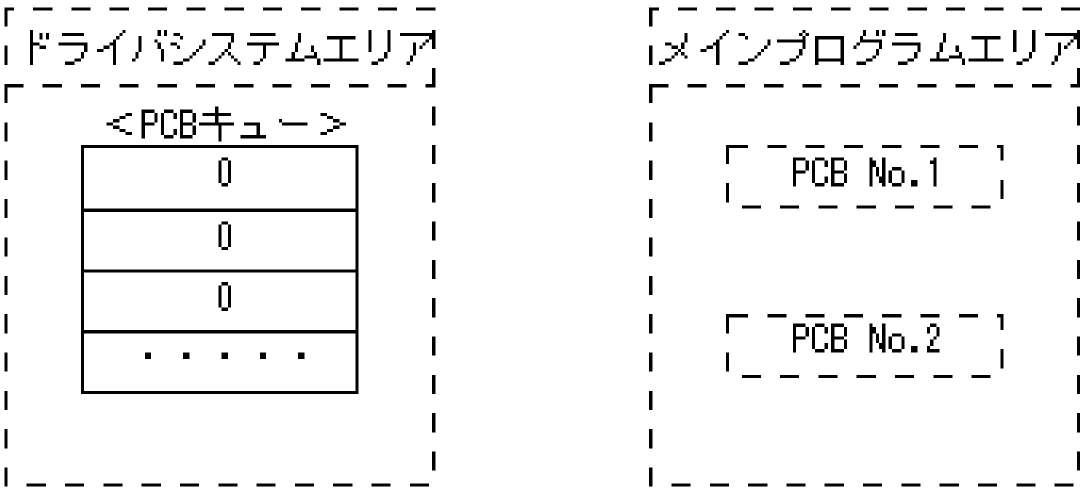
② ファンクションNo. 2の発行



③ ファンクションNo. 1の終了



④ ファンクションNo.2の終了



なお，PCBキューが一杯の状態で「PCM/ADPCMの録音/再生(AH=10H)」コマンドを実行するとキュービジーエラーを返す.

●録音/再生の連続実行

あるPCBに対する録音/再生処理が終了した後，次の(ペンディングになっていた)PCBの処理に移行する場合，以下の条件が満たされれば録音/再生は途切れずに継続される.

・両PCBの録音/再生サブコマンドが同一
・両PCBの録音/再生モードが同一

上の条件が満たされない場合には，前の録音/再生処理が一旦中断した後，新たな録音/再生が開始されます.

●録音/再生の強制終了

録音/再生中に以下のいずれかが起きた場合，録音/再生処理は強制終了する．強制終了の詳細については「録音/再生の強制終了(AH=11H)」を参照のこと．

・録音/再生中に以下のコマンドを実行した場合
　「録音/再生の強制終了(AH=11H)」
　「PCM/DSPドライバの初期化(AH=00H)」
・録音/再生中にEMSエラーが発生した場合
・録音/再生中にタイムアウトエラーが発生した場合
・録音/再生中に転送エラーが発生した場合

●PCMモードとデータフォーマット

PCMモードとバッファ上のデータフォーマットの関係を以下に示す．16ビットPCM録音では，データが上位バイト，下位バイトの順に格納される．

Lモノラル

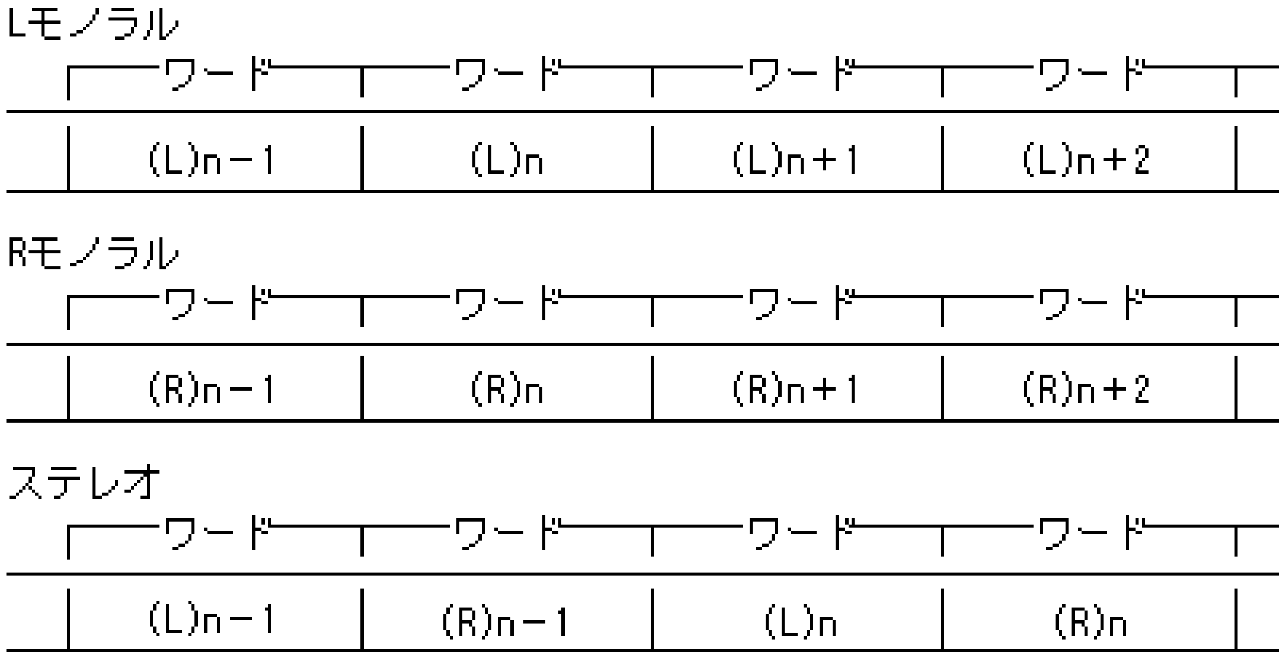
バイト	バイト	バイト	バイト
(L)n−1	(L)n	(L)n+1	(L)n+2

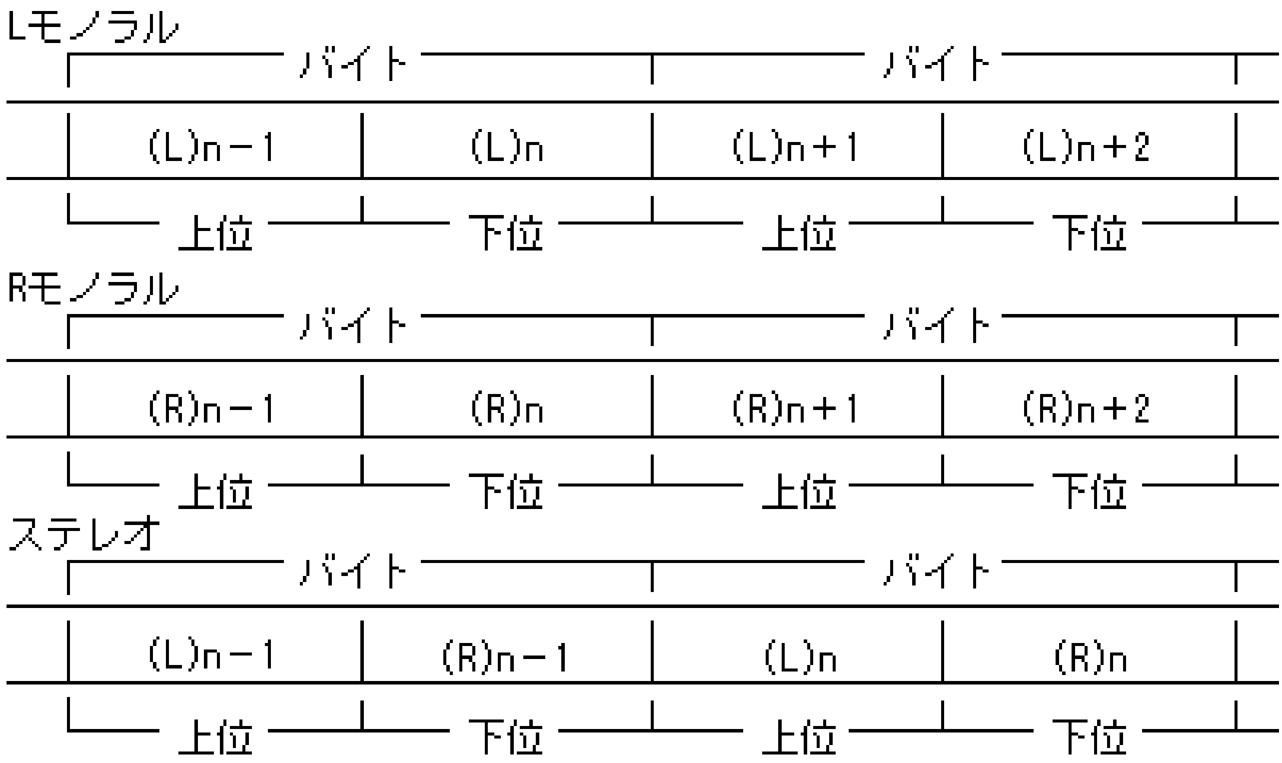
Rモノラル

バイト	バイト	バイト	バイト
(R)n−1	(R)n	(R)n+1	(R)n+2

ステレオ

バイト	バイト	バイト	バイト
(L)n−1	(R)n−1	(L)n	(R)n





●注意事項

① PCM/ADPCMの録音/再生(AH=10H)コマンドを用いてCD-DAの信号を録音することはできない．(GS/PC-9801-73のみ)

② ADPCM録音を指定した場合，LINE-OUT端子および内蔵スピーカーにはDSPの信号が出力されない．このため，音を出しながら録音を行いたい場合には VOL1/VOL3に0以外の値を設定する必要がある．(GS/PC-9801-73のみ)

③ PCBは，終了ステータスにFFH以外の値が返されるまで変更することはできない．

④ 録音/再生中に以下のコマンドを実行すると，録音/再生ビジーエラーを返す．

・ファンクションの初期化(AH=01H)
・パラメータの設定(AH=03H)
・ミキシングレートの設定(AH=07H)
・サンプリング周波数の設定(AH=0BH)
・PCMモードの設定(AH=0DH)

$STPTRN	AH=11H	
録音/再生の強制終了		

入　力

AH = 11H

出　力

Carry = 終了ステータス
　　　0 ： 正常終了
AH = 11H

機　能

　録音/再生処理を強制的に終了させる.
　まず現在の録音/再生処理を中断し，処理中およびペンディング中のすべてのPCBに強制終了ステータスを返した後，それらのPCBをキューから切り離す．したがって，コマンド終了時にはPCBのキューが空になる.
　なお，録音/再生中でない場合は何もせずにメインプログラムに戻る.

$SNSTRN	AH=12H	

録音/再生ステータスの取得

入　力

AH = 12H

出　力

Carry = 終了ステータス

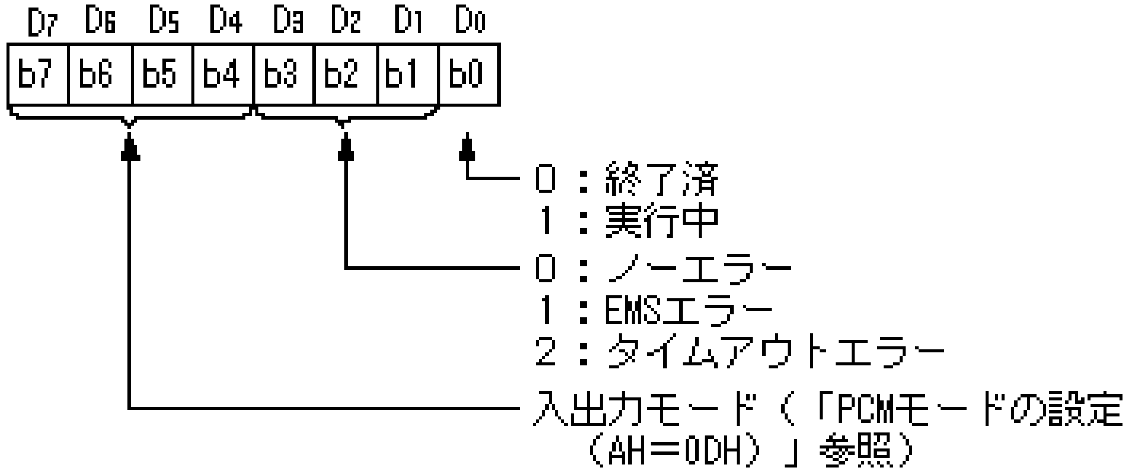
0 ： 正常終了

1 ： エラー

AH = エラーステータス(03H, 0FH)

正常終了時は12H

AL = 録音ステータス



CX:DX = 録音/再生済みサイズ

(CX×10000H+DX)

ES:BX = 実行中のPCBのアドレス(実行中のみ)

機　能

録音/再生時のステータスを取得する.

PCBの終了ステータスと異なり，このコマンドはALにコマンド実行時の実際の録音/再生状態を返す.

また，CX:DXには録音/再生待ちのデータサイズを返す．この値は，処理中のPCBの残りデータサイズと，以下の条件を満たすペンディング中の全PCBのバッファサイズを合計したものである.

・処理中のPCBとPCBモードが同一
・処理中のPCBとサンプリング周波数が同一
・過去に登録されたすべてのPCBが上記の2条件を満たす

複数バッファにまたがる録音/再生を連続で行う場合には，このCX:DXが0になる前に新たなPCM/ADPCMの録音/再生(AH=10H)を実行する必要がある.

第5章

DSPファンクション

5.1　概要

番号	名　称	内　容
00H	$Thru	スルー
01H	$Echo	エコー
02H	$LPF	ローパスフィルタ
03H	$HPF	ハイパスフィルタ
04H	$Geq	グラフィック・イコライザ
05H	$Rev	リバーブ
06H	$Hall	ホール
07H	$Loc	音像定位
08H	$Adj	定位補正
09H	$Pan	パンニング
0AH	$Chor	ステレオコーラス
0BH	$Doubl	ダブリング
0CH	$Flang	フランジャ
0DH	$Vibrt	ビブラート
0EH	$Ensem	アンサンブル
0FH	$Pitch	ピッチシフタ
10H	$Surr	サラウンド
11H	$Kara	疑似カラオケ
12H	$ADPCM	ADPCM 録音/再生

DSP上でファンクションを実行させる場合には，まず「ミュート状態の設定(AH=09H)」でミュートOFFの状態を確保した後，「ファンクションの初期化(AH=01H)」でファンクションを実行状態にする．

また，ファンクションの内部パラメータを変更したい場合には，「パラメータの設定(AH=03H)」を用いる．「パラメータの設定(AH=03H)」でのパラメータの受け渡しにはパラメータブロックを用いる．以下に各ファンクションのパラメータブロックの内訳を示す．

なお，ディレイ時間および周波数はすべて標準サンプリング44.1kHz時の値が示されている．他のサンプリングにおけるディレイ時間/周波数は，次の補正値κを用いて算出する(ディレイ時間は乗算，周波数は除算になる)．

サンプリング周波数	κ
44.10kHz	1.00
33.08kHz	1.33
22.05kHz	2.00
16.54kHz	2.67
11.03kHz	4.00
8.27kHz	5.33
5.52kHz	8.00
4.13kHz	10.68

5.2　ファンクション詳細

$Thru	00H	
スルー		

番　号

00H

パラメータ

オフセット	データ長	内　容	範囲
+0	バイト	フロントL出力	0
+1	バイト	リアL出力	0~100
+2	バイト	フロントR出力	〃
+3	バイト	リアR出力	〃

解　説

Lチャネル入力をフロントLとリアLに，Rチャネル入力をフロントRとリアRにスルーで出力する．

また，パラメータの変更により，フロント/リアの各々の出力レベルが変化する．

●フロントL～リアR出力

フロントL～リアRの出力レベルを指定する．出力レベルは0がゼロ出力を，100が最大出力を表す．

0～100(デフォルト100)

$Echo	01H	
エコー		

番　号

01H

パラメータ

●モノラル方式

オフセット	データ長	内　容	範囲
+0	バイト	エコー方式	0
+1	バイト	フロントL出力	0～100
+2	バイト	リアL出力	〃
+3	バイト	フロントR出力	〃
+4	バイト	リアR出力	〃
+5	バイト	L/R合成フィードバック	〃
+6	バイト	リザーブ	0
+7	バイト	L/R合成ディレイ	0～180
+8	バイト	リザーブ	0

●ステレオ方式

オフセット	データ長	内　容	範囲
+0	バイト	エコー方式	1
+1	バイト	フロントL出力	0～100
+2	バイト	リアL出力	〃
+3	バイト	フロントR出力	〃
+4	バイト	リアR出力	〃
+5	バイト	Lチャネルフィードバック	〃
+6	バイト	Rチャネルフィードバック	〃
+7	バイト	Lチャネルディレイ	0～180
+8	バイト	Rチャネルディレイ	〃

解　説

4チャネルのエコー制御を行う.

エコー方式にはモノラル方式とステレオ方式の2種類があり，パラメータブロックの第1バイトで選択する.

●エコー方式

モノラル方式とステレオ方式のいずれかを指定する

0…モノラル方式

4つのスピーカすべてに入力L/Rの合成信号を出力する. モノラル方式では個々のチャネルごとのエコー制御はできないが，ステレオ方式の倍のディレイ時間を設定することができる.

1…ステレオ方式

Lチャネル入力をフロントLとリアLスピーカに，Rチャネル入力をフロントRとリアRのスピーカにそれぞれ出力する. ステレオ方式ではL/Rの各チャネルごとにエコー制御を行うことができる.

(デフォルトは0)

●フロントL～リアR出力

フロントL～リアRの出力レベルを指定する．出力レベルは0がゼロ出力を，100が最大出力を表す．

0～100(デフォルト100)

●フィードバック

エコー信号の強さ(ディレイ信号のフィードバック量)を指定する．この値を0以外にすると出力にエコーがかかる．

0～100(デフォルトは100)

●ディレイ

各チャネルのディレイ時間を指定する．ディレイ時間とフィードバックに0を指定するとスルー出力になる．

モノラル：0～180(0～360×κms；デフォルト20×κms)
ステレオ：0～180(0～180×κms)

$LPF	02H	

ローパスフィルタ

番　号

02H

パラメータ

オフセット	データ長	内　容	範囲
+0	バイト	フロントＬ出力	0~100
+1	バイト	リアＬ出力	〃
+2	バイト	フロントＲ出力	〃
+3	バイト	リアＲ出力	〃
+4	バイト	Ｌチャネルカットオフ周波数	0~9
+5	バイト	Ｒチャネルカットオフ周波数	0~9

解 説

L/Rの各々について，高域成分のカットオフを行う.

●L/Rチャネルカットオフ周波数

高域=成分をカットする境界周波数を指定する.

0 … 200÷κHz
1 … 300÷κHz
2 … 500÷κHz
3 … 700÷κHz
4 … 850÷κHz
5 … 1.3÷κKHz
6 … 2÷κKHz
7 … 3÷κKHz
8 … 5÷κKHz
9 … 20÷κKHz
(デフォルト9)

●フロントL～リアR出力

フロントL～リアRの出力レベルを指定する. 出力レベルは0がゼロ出力を，100が最大出力を表す.

0～100(デフォルト100)

$HPF	03H	
ハイパスフィルタ		

番　号

03H

パラメータ

オフセット	データ長	内　容	範囲
+0	バイト	フロントL 出力	0～100
+1	バイト	リア L 出力	〃
+2	バイト	フロントR 出力	〃
+3	バイト	リア R 出力	〃
+4	バイト	L チャネル カットオフ周波数	0～7
+5	バイト	R チャネル カットオフ周波数	0～7

解　説

L/Rの各々について，低域成分のカットオフを行う．

●L/Rチャネルカットオフ周波数

低域成分をカットする境界周波数を指定する．

0 … 50÷κHz

1 … 100÷κHz

2 … 500÷κHz

3 … 1÷κKHz

4 … 3÷κKHz

5 … 5÷κKHz

6 … 7÷κKHz

7 … 10÷κKHz

（デフォルト0）

●フロントL～リアR出力

フロントL～リアRの出力レベルを指定する．出力レベルは0がゼロ出力を，100が最大出力を表す．

0～100（デフォルト100）

$Geq	04H	

グラフィック・イコライザ

番　号

04H

パラメータ

オフセット	データ長	内　容	範囲
+0	バイト	L チャネル 63 ÷ κ Hz 出力	0~100
+1	バイト	L チャネル 160 ÷ κ Hz 出力	〃
+2	バイト	L チャネル 400 ÷ κ Hz 出力	〃
+3	バイト	L チャネル 1.0 ÷ κ kHz 出力	〃
+4	バイト	L チャネル 2.5 ÷ κ kHz 出力	〃
+5	バイト	L チャネル 6.3 ÷ κ kHz 出力	〃
+6	バイト	L チャネル 12.5 ÷ κ kHz 出力	〃
+7	バイト	L チャネル スルー 出力	〃
+8	バイト	R チャネル 63 ÷ κ Hz 出力	〃
+9	バイト	R チャネル 160 ÷ κ Hz 出力	〃
+A	バイト	R チャネル 400 ÷ κ Hz 出力	〃
+B	バイト	R チャネル 1.0 ÷ κ kHz 出力	〃
+C	バイト	R チャネル 2.5 ÷ κ kHz 出力	〃
+D	バイト	R チャネル 6.3 ÷ κ kHz 出力	〃
+E	バイト	R チャネル 2.5 ÷ κ kHz 出力	〃
+F	バイト	R チャネル スルー 出力	〃

解　説

入力L/R信号の各々に対して，7バンドのイコライジング(周波数バンド別のレベル制御)を行う.

ファンクション内では，160Hz～12.5kHzの周波数帯に対応している.

●L/Rチャネル63÷κHz～12.5÷κkHz出力/スルー出力

L/Rチャネルの各々における各周波数バンドの出力レベルを指定する.

0～100(デフォルト30)

$Rev	05H	
リバーブ		

番　号

05H

パラメータ

オフセット	データ長	内　容	範囲
+0	バイト	リバーブレベル	0～100
+1	バイト	リバーブ時間	1～30

解　説

残響効果のうち，とくに後部残響音（リバーブ音）に重点を置いた音場生成を行う．

ホールと比較して，より響きのあるリバーブ音を生成させることができ，とくにテンポのゆるやかな曲の演奏時に効果を発揮する．

●リバーブレベル

後部=残響音の出力レベルを指定する．

0～100（デフォルト100）

●リバーブ時間

後部=残響音の持続時間を指定する．

1～30（0．1～3．0×κsec；デフォルト2．0×κ）

$Hall	06H	
ホール		

番　号

06H

パラメータ

オフセット	データ長	内　容	範囲
+0	バイト	初期ディレイ	1~50
+1	バイト	ルームサイズ	1~20
+2	バイト	ライブネス	0~10
+3	バイト	リバーブ時間	1~30
+4	バイト	リバース指定	0~1

解　説

入力L/R信号を元に初期反射音と後部残響音(リバーブ音)を生成し，それらを合成して音響ホールに似た音場を再現する．

初期反射音はホールの壁に1～2回反射して耳に届く音を，後部残響音とはホールの壁に何度も反射してから届く(方向性をもたない)音を意味する．ファンクションでは，L/Rの各々についてI～IVの4本の初期反射音をサポートしている．

●初期ディレイ
直接音から最初の初期反射音までの経過時間を指定する.

1～50(1～50×κms;デフォルト20×κ)

●ルームサイズ
初期反射音I～IVの相互の間隔を指定する.

1～20(1～20×κms;デフォルト15×κ)

●ライブネス
初期反射音の減衰速度(直接音に対して初期反射音I～IVがどのような比率で減衰していくか)を指定する.

0～10(デフォルト5)

●リバーブ時間
後部残響音の持続時間を指定する.

1～30(0.1～3.0×κsec;デフォルト2.0×κ)

●リバース指定
初期反射音の減衰をリバース状態にするかどうかを指定する.

0…ノーマル，1…リバース(デフォルトは0)

$Loc	07H	

音像定位

番　号

07H

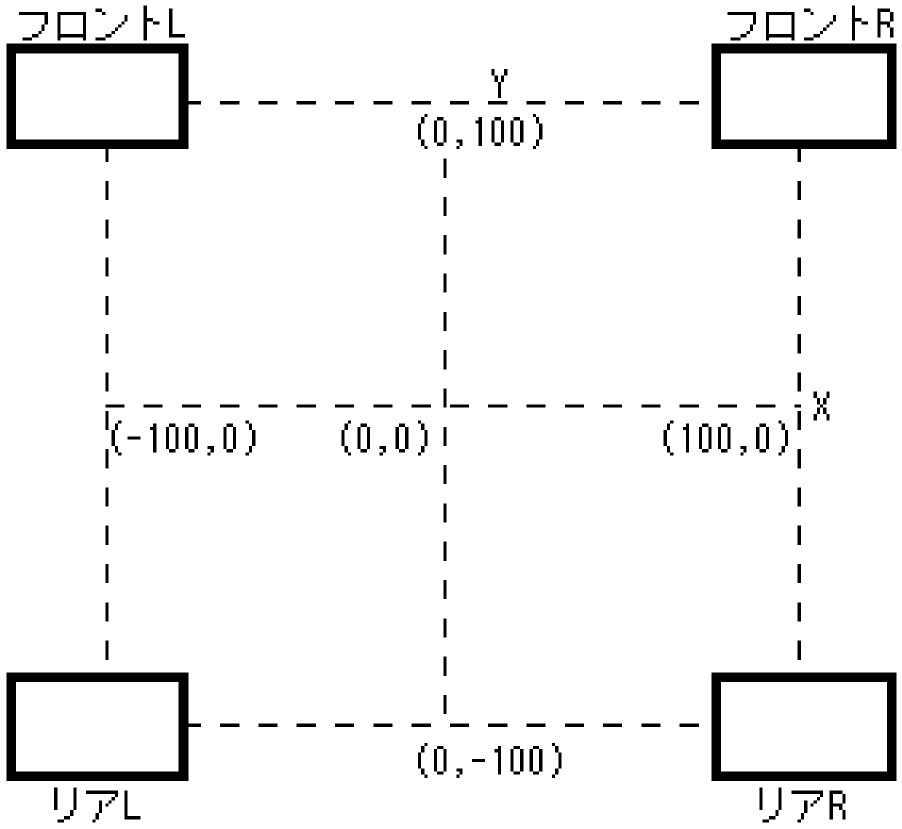
パラメータ

オフセット	データ長	内　容	範囲
+0	バイト	X座標	-100～100
+1	バイト	Y座標	-100～100

解　説

音像を(X，Y)座標で示される特定の位置に定位させる.

(X，Y)座標は，リスニング・ポイントを原点として以下のようにマッピングされる. この座標を逐次的に変化させることにより，任意の軌跡にそって音像を移動させることができる.



●座標系式

直行座標と極座標いずれかを指定する(デフォルトは0).

0 … 直交座標

音像位置を(X，Y)座標により指定する.

1 … 極座標

音像位置を（R，θ）座標により指定する.

●X/Y座標

-100～100(デフォルトは0)

●R座標

0～100(デフォルトは0)

●θ座標

0～180(0～360°；デフォルトは0)

$Adj	08H	

定位補正

番　号

08H

パラメータ

オフセット	データ長	内　容	範囲
+0	バイト	フロントL出力	0〜100
+1	バイト	リアL出力	〃
+2	バイト	フロントR出力	〃
+3	バイト	リアR出力	〃
+4	バイト	フロントLディレイ	0〜180
+5	バイト	リアLディレイ	〃
+6	バイト	フロントRディレイ	〃
+7	バイト	リアRディレイ	〃

解　説

4チャネル独立にディレイ時間の制御を行う．

4つのスピーカを長方形に配置できないような場合，その距離の違いから生じる位相差を補正するのに使用する．

●フロントL～リアR出力

フロントL～リアRの出力レベルを指定する．出力レベルは0がゼロ出力を，100が最大出力を表す．

0～100（デフォルト100）

●フロントL～リアRディレイ

フロントL～リアRのディレイ時間を指定する．リスニング・ポイントに近いスピーカほど大きな値にする．

0～180（0～180×κms；デフォルトは0）

$Pan	09H	

パンニング

番　号

09H

パラメータ

オフセット	データ長	内　容	範囲
+0	バイト	パンニング方式	0~3
+1	バイト	パンニング周波数	1~100

解　説

音をL/Rまたはフロント/リア間で往復させたり，4スピーカ間で回転させたりする.

●パンニング方式

音の移動方式(L/R往復，フロント/リア往復等)を指定する.

0 … 右回転
1 … 左回転
2 … L/R往復
3 … フロント/リア往復
(デフォルトは0)

●パンニング周波数

往復/回転の周波数を指定する.

1〜100(0.1〜10.0÷κHz;デフォルト1.0)

$Chor	0AH	
ステレオコーラス		

入　力

0AH

パラメータ

オフセット	データ長	内　容	範囲
+0	バイト	フロントL出力	0~100
+1	バイト	リアL出力	〃
+2	バイト	フロントR出力	〃
+3	バイト	リアR出力	〃
+4	バイト	ディレイL出力	〃
+5	バイト	ディレイR出力	〃
+6	バイト	変調中心	0~250
+7	バイト	変調周波数	1~100
+8	バイト	変調レベル	0~100

解　説

パラメータの調整により，ステレオ効果/コーラス効果/トレモロ効果の3種類の効果を実現する．

ステレオ効果 … モノラルソースにステレオ効果を付加して出力する．
変調周波数 0.2Hz付近
変調レベル 1.0ms付近
コーラス効果 … 音に厚みをもたせて出力する．
変調周波数 0.5Hz付近
変調レベル 5.0ms付近
トレモロ効果 … 回転スピーカに似た音を生成する．
変調周波数 5.0Hz付近
変調レベル 1.0ms付近

●フロントL～リアR出力
フロントL～リアRの出力レベルを指定する．

0～100(デフォルトは100)

●ディレイL/R出力
ディレイ信号のフロント・リアL/フロント・リアRへの出力レベルを指定する．

0～100(デフォルトは100)

●変調中心

ディレイ時間の変調中心を指定する．ディレイ時間は，この点を中心として周期的な増を繰り返す．

0〜250(0.0〜50.0×κms;デフォルト1.0×κ)

●変調周波数

ディレイ時間の変調周波数を指定する．

1〜100(0.1〜10.0÷κHz;デフォルト1.0÷κ)

●変調レベル

ディレイ時間の変調振幅を指定する．

0〜100(0.0〜10.0×κms;デフォルト1.0×κ)

$Doubl	0BH	
ダブリング		

番　号

0BH

パラメータ

オフセット	データ長	内　容	範囲
+0	バイト	フロントL出力	0~100
+1	バイト	リアL出力	〃
+2	バイト	フロントR出力	〃
+3	バイト	リア R出力	〃
+4	バイト	ディレイL出力	〃
+5	バイト	ディレイR出力	〃
+6	バイト	ディレイ時間	0~180

解　説

音がにじんだような効果を作り出す.

●フロントL～リアR出力

フロントL～リアRの出力レベルを指定する.

0～100(デフォルトは100)

●ディレイL/R出力

ディレイ信号のフロント・リアL/フロント・リアRへの出力レベルを指定する.

0～100(デフォルトは100)

●ディレイ時間

ディレイ信号のスルー信号に対するディレイ時間を指定する.

0～180(0～180×κms;デフォルト100×κ)

$Flang	0CH	

フランジャ

番　号

0CH

パラメータ

オフセット	データ長	内　容	範囲
+0	バイト	フロントL出力	0～100
+1	バイト	リアL出力	〃
+2	バイト	フロントR出力	〃
+3	バイト	リアR出力	〃
+4	バイト	ディレイL出力	〃
+5	バイト	ディレイR出力	〃
+6	バイト	変調中心	0～250
+7	バイト	変調周波数	1～100
+8	バイト	変調レベル	0～100
+9	バイト	フィードバック	〃

解　説

うねりのある，メタリックな音を作り出す．

●フロントL～リアR出力

フロントL～リアRの出力レベルを指定する．

0～100(デフォルトは100)

●ディレイL/R出力

ディレイ信号のフロント・リアL/フロント・リアRへの出力レベルを指定する．

0～100(デフォルトは100)

●変調中心

ディレイ時間の変調中心を指定する．ディレイ時間は，この点を中心として周期的な増を繰り返す．

0～250(0.0～50.0×κms;デフォルト2.8×κ)

●変調周波数

ディレイ時間の変調周波数を指定する．

1～100(0.1～10.0÷κHz;デフォルト1.0÷κ)

●変調レベル
ディレイ時間の変調振幅を指定する.

0〜100(0.0〜10.0×κms;デフォルト1.0×κ)

●フィードバック
ディレイ信号のフィードバック量を指定する．フィードバックを強くかけると，メタリック感が強調される.

0〜100(デフォルト30)

$Vibrt	0DH	
ビブラート		

番　号

0DH

パラメータ

オフセット	データ長	内　容	範囲
+0	バイト	フロントL出力	0〜100
+1	バイト	リアL出力	〃
+2	バイト	フロントR出力	〃
+3	バイト	リア R出力	〃
+4	バイト	ディレイL出力	〃
+5	バイト	ディレイR出力	〃
+6	バイト	変調中心	0〜250
+7	バイト	変調周波数	1〜100
+8	バイト	変調レベル	0〜100

解　説

スルー成分をカットし，変調成分のみを出力する.

●フロントL～リアR出力

フロントL～リアRの出力レベルを指定する.

0～100(デフォルトは100)

●ディレイL/R出力

ディレイ信号のフロント・リアL/フロント・リアRへの出力レベルを指定する.

0～100(デフォルトは100)

●変調中心

ディレイ時間の変調中心を指定する. ディレイ時間は，この点を中心として周期的な増を繰り返す.

0～250(0.0～50.0×κms;デフォルト1.0×κ)

●変調周波数

ディレイ時間の変調周波数を指定する.

1～100(0.1～10.0÷κHz;デフォルト1.0÷κ)

●変調レベル

ディレイ時間の変調振幅を指定する.

0～100(0.0～10.0×κms;デフォルト1.0×κ)

$Ensem	0EH	
アンサンブル		

番　号

0EH

パラメータ

オフセット	データ長	内　容	範囲
+0	バイト	フロントL出力	0～100
+1	バイト	リアL出力	〃
+2	バイト	フロントR出力	〃
+3	バイト	リアR出力	〃
+4	バイト	ディレイ1 L出力	〃
+5	バイト	ディレイ1 R出力	〃
+6	バイト	ディレイ1 変調中心	0～250
+7	バイト	ディレイ1 変調周波数	1～100
+8	バイト	ディレイ1 変調レベル	0～100
+9	バイト	ディレイ2 L出力	〃
+A	バイト	ディレイ2 R出力	〃
+B	バイト	ディレイ2 変調中心	0～250
+C	バイト	ディレイ2 変調周波数	1～100
+D	バイト	ディレイ2 変調レベル	0～100

解　説

2つのディレイ要素により，多人数で演奏しているような効果を作り出す.

●フロントL～リアR出力
フロントL～リアRの出力レベルを指定する.

0～100(デフォルトは100)

● ディレイ1 L/R出力
ディレイ1信号のフロント・リアL/フロント・リアRへの出力レベルを指定する.

0～100(デフォルト100)

●ディレイ1 変調中心
ディレイ1の変調中心を指定する．ディレイ1の遅延時間は，この点を中心として周期的な増減を繰り返す.

0～250(0.0～50.0×κms;デフォルト1.0×κ)

●ディレイ1 変調周波数
ディレイ1の変調周波数を指定する.

1～100(0.1～10.0÷κHz;デフォルト0.5÷κ)

●ディレイ1 変調レベル
ディレイ1の変調振幅を指定する.

0〜100(0.0〜10.0×κms;デフォルト1.0×κ)

●ディレイ2 L/R出力
ディレイ1の場合と同一.

●ディレイ2 変調中心
ディレイ1の場合と同一.

●ディレイ2 変調周波数
ディレイ2の変調周波数を指定する.

1〜100(0.1〜10.0÷κHz;デフォルト5.0÷κ)

●ディレイ2 変調レベル
ディレイ1の場合と同一.

$Pitch	0FH	

ピッチシフタ

番　号

0FH

パラメータ

オフセット	データ長	内　容	範囲
+0	バイト	L チャネル ピッチ	-120～120
+1	バイト	R チャネル ピッチ	-120～120

解　説

音のピッチ(音程)を変化させて出力する.

●L/Rチャネルピッチ

L/R各々のピッチを，10セント(半音の1/10)を単位として±1オクターブの範囲で指定する．ピッチは2の補数表現で表し，プラスが高音側へのシフトを，マイナスが低音側へのシフトを意味する.

-120～120(-1～+1オクターブ;デフォルトは0)

注　意

ピッチシフタの場合，サンプリング周波数を下げても音質が下がるだけで効果は何もないため，標準のサンプリングで動作させるのが理想的である.

$Surr	10H	

サラウンド

番　号

10H

パラメータ

オフセット	データ長	内　容	範囲
+0	バイト	フロントLディレイ	0～90
+1	バイト	リアLディレイ	0～90
+2	バイト	フロントRディレイ	0～90
+3	バイト	リアRディレイ	0～90
+4	バイト	フロントLミックス	0～50
+5	バイト	リアLミックス	0～50
+6	バイト	フロントRミックス	0～50
+7	バイト	リアRミックス	0～50

解　説

リスナーを包み込むような音場を生成する.

●フロントL～リアRディレイ
フロントL～リアRのディレイ時間を指定する.

0～90(0～90×κms;デフォルトは50)

●フロントL/リアLミックス
フロントL/リアL出力における，L入力成分に対するR入力成分のミックスレートを指定する.

0～50%(デフォルトは50)

●フロントR/リアRミックス
フロントR/リアR出力における，R入力成分に対するL入力成分のミックスレートを指定する.

0～50%(デフォルトは50)

$Kara	11H	
疑似カラオケ		

番　号

11H

パラメータ

オフセット	データ長	内　容	範囲
+0	バイト	フロントL出力	0～100
+1	バイト	リアL出力	〃
+2	バイト	フロントR出力	〃
+3	バイト	リアR出力	〃

解　説

ソース信号中の音声信号を除去してから出力する．ただし，ステレオ録音されたソースについてのみ有効となる．

●フロントL～リアR出力

フロントL～リアRの出力レベルを指定する．

0～100（デフォルトは100）

$ADPCM	12H	

ADPCM録音/再生

番　号

12H

パラメータ

オフセット	データ長	内　容	範囲
+0	バイト	フロントL出力	0～100
+1	バイト	リアL出力	〃
+2	バイト	フロントR出力	〃
+3	バイト	リアR出力	〃

機　能

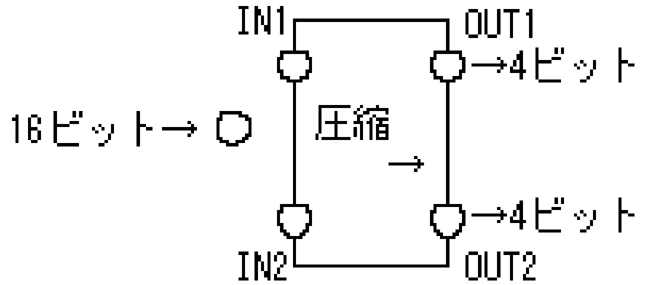
ADPCM録音/再生を行う.

入出力モード(「PCMモードの設定(AH=0DH)」参照)が入力に設定されている場合にはADPCM録音が,出力に設定されている場合にはADPCM再生が選択される.

ADPCM録音は,16ビットサウンドデータを帯域圧縮して4ビットのデータに変換する.またADPCM再生は,この圧縮されたデータを元のサウンドデータに展開する.

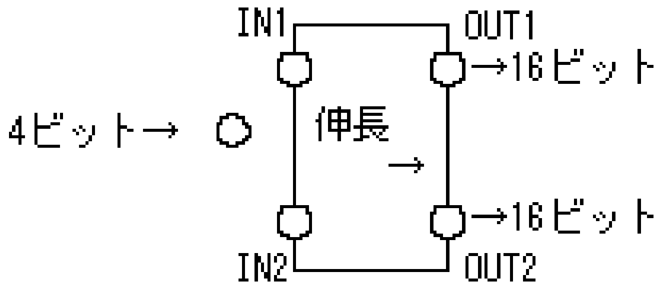
[録音時]

（入力時）



[再生時]

（出力時）



注　意

ADPCM録音時の入力端子にはOUT1/OUT2のいずれかを指定する必要がある．IN1/IN2を指定すると，圧縮前のデータが取得される．

ADPCM録音時には，LINE-OUT端子への出力は自動的にミュートされる．これは帯域圧縮したデータをそのままスピーカに出力すると雑音になるためである．

ADPCM再生時には，CD-DAおよびFM+LINE-INからの入力がミュートされる．これは通常のPCMデータに展開処理を施すと雑音になるためである．

ADPCMは録音/再生専用のファンクションであるため，入出力モード0で実行させることはできない．モード0でADPCMの「ファンクションの初期化(AH=01H)」コマンドを実行するとエラーになる．同様に，ADPCMがロードされているときにモード0の「PCMモードの設定(AH=0DH)」コマンドを発行するとエラーになる．

付録

第1章

サンプルプログラム

1.1　サンプルプログラム利用上の注意

本編に掲載されているサンプルプログラムは，Microsoft C Ver.7でコンパイルされることを前提とした，C言語によるソースファイルです．nmakeコマンドによりメイクファイルを利用してコンパイルして下さい．

1.2　AVGDRV.SYS サンプルプログラム

■メイクファイル(MAKEFILE)

```
CC=cl
OBJECTS=start.obj stop.obj ¥
        stat.obj show.obj ¥
        color.obj mono.obj ¥
        double.obj half.obj ¥
        full.obj mask.obj ¥
        bright.obj dark.obj ¥
        videodrv.obj
PROGRAMS=start.exe stop.exe ¥
         stat.exe show.exe ¥
         color.exe mono.exe ¥
         double.exe half.exe ¥
         full.exe mask.exe ¥
         bright.exe dark.exe
INCPATH=..
CFLAGS=-I$(INCPATH) -Zi -G2 -nologo
LDFLAGS=$(CFLAGS) -F 1000
VPATH=$(INCPATH)

all : $(PROGRAMS)

$(PROGRAMS) : videodrv.obj

$(OBJECTS) : video.h

.obj.exe :
    $(CC) $(LDFLAGS) -o $@ $* videodrv
```

```
.c.obj :
    $(CC) $(CFLAGS) -c $*.c
```

■ヘッダファイル(VIDEO.H)

```
/*
 * type definition
 */
typedef enum { FALSE,  TRUE }    BOOLEAN;
typedef enum { SUCCESS,  FAIL }  RESULT;

/*
 * macro
 */
#define CALLDRV(i,c,r)   {struct _SREGS s;¥
                         _segread(&s);r.h.ah=(char)c;¥
                         _int86x(i,&r,&r,&s);¥
                         }
/*
 * constant
 */
#define DEVGRDRV        "AVGDRV$$"  /* name of the driver */

#define INTGRDRV        0xd8        /* interrupt number */

#define CMDSETSYNC      0x00
#define CMDALLOCMODE    0x02
#define CMDINITFRAME    0x04
#define CMDSETFRAME     0x05
#define CMDASKFRAME     0x06
#define CMDCOMPOSE      0x07
#define CMDSETMASK      0x0c
#define CMDSTART        0x19
#define CMDEND          0x1a
#define CMDGETINFO      0x1b
#define CMDDIGITIZE     0x20
#define CMDV_EFECT      0x24
```

```
#define CMDHUE          0x26
#define CMDCOLOR        0x27
#define CMDBRIGHT       0x28
#define CMDCONTRAST     0x29
#define CMDASKLEVEL     0x2a
#define CMDV_WINDOW     0x2c
#define CMDV_SIZE       0x2d

/*
 * prototype
 */
RESULT  start_videodrv ( void );
void    end_videodrv ( void );
int     call_videodrv ( int, union _REGS * );
```

■ドライバを呼び出す(VIDEODRV.C)

```
#include <stdio.h>
#include <dos.h>

#include "video.h"

RESULT  start_videodrv ( void ) {
    union   _REGS   regs;
    int     fh;

    /* check driver is available or not ? */
    if ( ( fh = open ( DEVGRDRV, 0 ) ) == EOF ) {
        return FAIL;
    }
    close ( fh );

    call_videodrv ( CMDSTART, &regs );
/*
 *  atexit ( end_videodrv );
 */
    return SUCCESS;
}

void    end_videodrv ( void ) {
    union   _REGS   regs;

    call_videodrv ( CMDEND, &regs );
}

int call_videodrv ( int cmd, union _REGS *regs ) {
    struct _SREGS   sregs;

    _segread ( &sregs );
```

```
    sregs.es = sregs.ds;
    regs->h.ah = cmd;
    _int86x ( INTGRDRV, regs, regs, &sregs );
    return regs->x.cflag;
}
```

■ドライバを起動する(START.C)

```
#include <stdio.h>
#include <dos.h>

#include "video.h"

void    main ( void ) {
    union _REGS regs;

    if ( start_videodrv ( ) ) {
        fprintf ( stderr, "Cannot start video driver (%x)¥n",
                  regs.h.ah );
        exit ( 1 );
    }
    regs.h.al = 0x08;
    regs.h.bh = 0x21;
    regs.h.bl = 0x01;
    if ( call_videodrv ( CMDSETSYNC, &regs ) ) {
        fprintf ( stderr, "Cannot set sync mode (%x)¥n",
                  regs.h.ah );
        exit ( 1 );
    }
    regs.h.al = 0x40;
    regs.h.bh = 0x09;
    regs.h.bl = 0x01;
    regs.h.ch = 0x01;
    regs.h.cl = 0x01;
    if ( call_videodrv ( CMDALLOCMODE, &regs ) ) {
        fprintf ( stderr, "Cannot set mode (%x)¥n",
                  regs.h.ah );
        exit ( 1 );
    }
    if ( call_videodrv ( CMDINITFRAME, &regs ) ) {
```

```
            fprintf ( stderr, "Cannot set mode (%x)¥n",
                      regs.h.ah );
            exit ( 1 );
        }
}
```

■ドライバを停止する(STOP.C)

```
#include <stdio.h>
#include <dos.h>

#include "video.h"

void    main ( void ) {
    union _REGS regs;

    end_videodrv ( );
    exit ( 0 );
}
```

■ドライバの状態を表示する(STAT.C)

```
#include <stdio.h>
#include <dos.h>
#include <malloc.h>

#include "video.h"

/*
 * type definition
 */
typedef struct {
    int version,
        mode,
        reserve0,
        extention,
        reserve1,
        overlay,
        pallete,
        display;
}    DATATABLE;

/*
 * constant
 */
const char * const StrDisplayMode[] = {
    "monochrome",
    "3-plane-color",
    "4-plane-color",
    "8-plane-color",
    "16-plane-color",
    "24-plane-color",
    "32-plane-color",
    "4bits/pix-color",
```

```
    "8bits/pix-color",
    "16bits/pix-color",
    "24bits/pix-color",
    "32bits/pix-color"
};

void    main ( int argc, char **argv ) {
    union _REGS  regs;
    DATATABLE    *table;
    int fh, i, mask;

    /* get table size */
    regs.h.al = 0x00;
    if ( call_videodrv ( CMDGETINFO, &regs ) ) {
        fprintf ( stderr, "Cannot get table size(%x)¥n",
                  regs.h.ah );
        exit ( 1 );
    }

    /* allocate table buffer */
    table = (DATATABLE *)malloc ( regs.x.cx );

    /* get information */
    regs.h.al = 0x01;
    regs.x.bx = (int)table;
    if ( call_videodrv ( CMDGETINFO, &regs ) ) {
        fprintf ( stderr, "Cannot get information(%x)¥n",
                  regs.h.ah );
        exit ( 1 );
    }

    /* show contents of the table */
    printf ( "AV Graphic driver¥n" );
    printf ( "version: %d¥n", table->version );
```

```
    printf ( "support mode:" );
    for ( i = 0, mask = 1; mask; ++i, mask <<= 1 ) {
        if ( table->mode & mask ) {
            printf ( " %s", StrDisplayMode[i] );
        }
    }
    printf ( "¥n" );
    printf ( "number of overlay: %d¥n", table->overlay );
    printf ( "number of pallette: %d¥n", table->pallete );
    printf ( "number of displays: %d¥n", table->display );

    exit ( 0 );
}
```

■表示を開始する(SHOW.C)

```
#include <stdio.h>
#include <dos.h>

#include "video.h"

int main ( void ) {
    union _REGS regs;

    regs.h.al = 0;
    regs.h.bh = 0x16;
    regs.h.bl = 0;
    regs.h.ch = 0;
    if ( call_videodrv ( CMDCOMPOSE, &regs ) ) {
        fprintf ( stderr, "Cannot compose (%x)¥n", regs.h.ah );
        exit ( 1 );
    }
    exit ( 0 );
}
```

■画面の一部をマスクする(MASK.C)

```
#include <stdio.h>
#include <dos.h>

#include "video.h"

int main ( void ) {
    union _REGS regs;
    int width, height;

    regs.h.al = 1;
    if ( call_videodrv ( CMDV_SIZE, &regs ) ) {
        fprintf ( stderr, "Cannot get size (%x)¥n",
                  regs.h.ah );
        exit ( 1 );
    }
    width = regs.x.bx;
    height = regs.x.cx;

    regs.h.al = 0xbf;
    regs.x.bx = width / 4;
    regs.x.cx = width * 3 / 4;
    regs.h.dh = (long)height * 200L / 480L / 4 + 20;
    regs.h.dl = (long)height * 200L / 480L * 3 / 4 + 20;
    fprintf ( stderr, "mask: (%d,%d)-(%d,%d)¥n",
        regs.x.bx, regs.h.dh, regs.x.cx, regs.h.dl );
    if ( call_videodrv ( CMDSETMASK, &regs ) ) {
        fprintf ( stderr, "Cannot set mask (%x)¥n",
                  regs.h.ah );
        exit ( 1 );
    }
    exit ( 0 );
}
```

■マスクを外す(FULL.C)

```
#include <stdio.h>
#include <dos.h>

#include "video.h"

int main ( void ) {
    union _REGS regs;
    int width, height;

    regs.h.al = 1;
    if ( call_videodrv ( CMDV_SIZE, &regs ) ) {
        fprintf ( stderr, "Cannot get size (%x)¥n",
                  regs.h.ah );
        exit ( 1 );
    }
    width = regs.x.bx;
    height = regs.x.cx;

    regs.h.al = 0x00;
    regs.x.bx = 0;
    regs.x.cx = width;
    regs.h.dh = 0;
    regs.h.dl = height;
    if ( call_videodrv ( CMDSETMASK, &regs ) ) {
        fprintf ( stderr, "Cannot set mask (%x)¥n",
                  regs.h.ah );
        exit ( 1 );
    }
    exit ( 0 );
}
```

■モノクロで表示する(MONO.C)

```
#include <stdio.h>
#include <dos.h>

#include "video.h"

int main ( void ) {
    union _REGS regs;

    regs.h.al = 0x1f;
    regs.h.bh = 255;
    regs.h.bl = 0;
    regs.h.ch = 127;
    regs.h.cl = -128;
    regs.h.dh = 127;
    regs.h.dl = -128;
    if ( call_videodrv ( CMDV_EFECT, &regs ) ) {
        fprintf ( stderr, "Cannot make effect (%x)¥n",
                  regs.h.ah );
        exit ( 1 );
    }

    exit ( 0 );
}
```

■カラーで表示する(COLOR.C)

```
#include <stdio.h>
#include <dos.h>

#include "video.h"

int main ( void ) {
    union _REGS regs;

    regs.h.al = 0x0f;
    regs.h.bh = 255;
    regs.h.bl = 0;
    regs.h.ch = 127;
    regs.h.cl = -128;
    regs.h.dh = 127;
    regs.h.dl = -128;
    if ( call_videodrv ( CMDV_EFECT, &regs ) ) {
        fprintf ( stderr, "Cannot make effect (%x)¥n",
                  regs.h.ah );
        exit ( 1 );
    }

    exit ( 0 );
}
```

■画面を明るくする(BRIGHT.C)

```
#include <stdio.h>
#include <dos.h>

#include "video.h"

int main ( void ) {
    union _REGS regs;
    int hue, color, bright, contrast;

    if ( call_videodrv ( CMDASKLEVEL, &regs ) ) {
        fprintf ( stderr, "Cannot get size (%x)¥n",
                  regs.h.ah );
        exit ( 1 );
    }
    hue = (char)regs.h.bh;
    color = (char)regs.h.bl;
    bright = (char)regs.h.ch;
    contrast = (char)regs.h.cl;

    regs.h.al = bright + 0x10;
    if ( call_videodrv ( CMDBRIGHT, &regs ) ) {
        fprintf ( stderr, "Cannot change brightness(%x)¥n",
                  regs.h.ah );
        exit ( 1 );
    }
    exit ( 0 );
}
```

■画面を暗くする(DARK.C)

```
#include <stdio.h>
#include <dos.h>

#include "video.h"

int main ( void ) {
    union _REGS regs;
    int hue, color, bright, contrast;

    if ( call_videodrv ( CMDASKLEVEL, &regs ) ) {
        fprintf ( stderr, "Cannot get size (%x)¥n",
                  regs.h.ah );
        exit ( 1 );
    }
    hue = (char)regs.h.bh;
    color = (char)regs.h.bl;
    bright = (char)regs.h.ch;
    contrast = (char)regs.h.cl;

    regs.h.al = bright - 0x10;
    if ( call_videodrv ( CMDBRIGHT, &regs ) ) {
        fprintf ( stderr, "Cannot change brightness(%x)¥n",
                  regs.h.ah );
        exit ( 1 );
    }
    exit ( 0 );
}
```

■表示サイズを倍にする(DOUBLE.C)

```
#include <stdio.h>
#include <dos.h>

#include "video.h"

typedef struct {
    int dx, dy,
        vw, vh,
        vx, vy;
}   WINDOW;

int main ( void ) {
    union _REGS regs;
    WINDOW  win;
    int width, height;

    regs.h.al = 1;
    if ( call_videodrv ( CMDV_SIZE, &regs ) ) {
        fprintf ( stderr, "Cannot get size (%x)¥n",
                  regs.h.ah );
        exit ( 1 );
    }
    width = regs.x.bx;
    height = regs.x.cx;
    fprintf ( stderr, "%d x %d ->", width, height );

    width *= 2;
    height *= 2;
    fprintf ( stderr, "%d x %d¥n", width, height );
    regs.h.al = 0;
    regs.x.bx = width;
    regs.x.cx = height;
```

```
    if ( call_videodrv ( CMDV_SIZE, &regs ) ) {
        fprintf ( stderr, "Cannot scale (%x)¥n",
                  regs.h.ah );
        exit ( 1 );
    }

    win.dx = win.dy = 0;
    win.vx = win.vy = 0;
    win.vw = width;
    win.vh = height;
    regs.h.al = 0;
    regs.x.bx = (int)&win;
    if ( call_videodrv ( CMDV_WINDOW, &regs ) ) {
        fprintf ( stderr, "Cannot set window (%x)¥n",
                  regs.h.ah );
        exit ( 1 );
    }

    exit ( 0 );
}
```

■表示サイズを半分にする(HALF.C)

```
#include <stdio.h>
#include <dos.h>

#include "video.h"

typedef struct {
    int dx, dy,
        vw, vh,
        vx, vy;
}   WINDOW;

int main ( void ) {
    union _REGS regs;
    WINDOW  win;
    int width, height;

    regs.h.al = 1;
    if ( call_videodrv ( CMDV_SIZE, &regs ) ) {
        fprintf ( stderr, "Cannot get size (%x)¥n",
                  regs.h.ah );
        exit ( 1 );
    }
    width = regs.x.bx;
    height = regs.x.cx;
    fprintf ( stderr, "%d x %d ->", width, height );

    width /= 2;
    height /= 2;
    fprintf ( stderr, "%d x %d¥n", width, height );
    regs.h.al = 0;
    regs.x.bx = width;
    regs.x.cx = height;
```

```
    if ( call_videodrv ( CMDV_SIZE, &regs ) ) {
        fprintf ( stderr, "Cannot scale (%x)¥n",
                    regs.h.ah );
        exit ( 1 );
    }

    win.dx = win.dy = 0;
    win.vx = win.vy = 0;
    win.vw = width;
    win.vh = height;
    regs.h.al = 0;
    regs.x.bx = (int)&win;
    if ( call_videodrv ( CMDV_WINDOW, &regs ) ) {
        fprintf ( stderr, "Cannot set window (%x)¥n",
                    regs.h.ah );
        exit ( 1 );
    }

    exit ( 0 );
}
```

■表示ウィンドウの大きさを表示する(WIN.C)

```
#include <stdio.h>
#include <dos.h>

#include "video.h"

int main ( void ) {
    union _REGS regs;
    struct _SREGS   sregs;
    int width, height;
    int far *p;

    _segread ( &sregs );
    sregs.es = sregs.ds;
    regs.h.ah = CMDV_WINDOW;
    regs.h.al = 1;
    _int86x ( INTGRDRV, &regs, &regs, &sregs );

    FP_OFF ( p ) = regs.x.bx;
    FP_SEG ( p ) = sregs.es;
    fprintf ( stderr, "(%d,%d)¥n", *p, *( p + 1 ) );
    fprintf ( stderr, "(%d,%d)+(%d,%d)¥n",
        *( p + 4 ), *( p + 5 ), *( p + 2 ), *( p + 3 ) );
    exit ( 0 );
}
```

1.3　AVSDRV.SYS サンプルプログラム

■メイクファイル(MAKEFILE)

```
CC=cl
INCPATH=..
OBJECTS=init.obj show.obj vlm.obj mix.obj mute.obj ¥
        freq.obj bit.obj chan.obj mode.obj¥
        rec.obj play.obj
PROGRAMS=init.exe show.exe vlm.exe mix.exe mute.exe ¥
         freq.exe bit.exe chan.exe mode.exe¥
         rec.exe play.exe
CFLAGS=-I$(INCPATH) -Zi -G2 -nologo
LDFLAGS=$(CFLAGS) -F 1000
VPATH=$(INCPATH)

all : $(PROGRAMS)

$(PROGRAMS) : pcmdrv.obj

.obj.exe :
    $(CC) $(LDFLAGS) -o $@ $* pcmdrv

$(OBJECTS) : pcm.h
```

■ヘッダファイル(PCM.H)

```
/*
 * type definition
 */
typedef enum { FALSE,  TRUE }     BOOLEAN;
typedef enum { SUCCESS, FAIL }  RESULT;

/*
 * macro
 */
#define CALLDRV(i,c,r)   {struct _SREGS s;¥
                        _segread(&s);r.h.ah=(char)c;¥
                        _int86x(i,&r,&r,&s);¥
                        }
/*
 * constant
 */
#define DEVSNDDRV   "AVSDRV$$"  /* name of the driver */

#define INTSNDDRV   0xda    /* interrupt  number for pcm */
#define INTPCMDRV   0xd9    /* interrupt  number for fm/ssg */

#define CMDINITDRV  0       /* pcm command */
#define CMDSETVOL   5
#define CMDGETVOL   6
#define CMDSETMIX   7
#define CMDGETMIX   8
#define CMDSETMUTE  9
#define CMDGETMUTE  10
#define CMDSETFREQ  11
#define CMDGETFREQ  12
#define CMDSETPCM   13
#define CMDGETPCM   14
```

```
#define CMDPCMTRN   16
#define CMDSTPTRN   17
#define CMDSNSTRN   18

#define CMDINITIALIZE2  3       /* fm/ssg command */
#define CMDCLEAR        2

#define MODE8BIT    0       /* sampling mode */
#define MODE16BIT   0x80
#define MODEMONOL   0x00
#define MODEMONOR   0x10
#define MODESTEREO  0x20

#define MEMMAIN     0       /* memory type for play/rec. */
#define MEMEMS      1

#define FRQ4410     0       /* play/rec. freq. */
#define FRQCD       FRQ4410
#define FRQ3308     1
#define FRQ2205     2
#define FRQ1654     3
#define FRQ1103     4
#define FRQ0827     5
#define FRQ0552     6
#define FRQ0413     7

#define ERRPARAM    0x02        /* error code to play/rec. */
#define ERREMS      0x03
#define ERRHNDL     0x04
#define ERRMEM      0x05
#define ERRQUEBUSY  0x06
#define ERRTIMEOUT  0x0f

#define PCMDONE     0       /* sampling/rec. status */
```

```
#define PCMABORT    1
#define PCMEMSERR   2
#define PCMTIMEOUT  3
#define PCMTRNSERR  4
#define PCMRUNNING  (-1)

/*
 * type definition
 */
typedef union pcmbuff {
    char    __far   *addr;
    unsigned long   offset;
}   pcmbuff;
typedef struct PCMPCB {
    char    mode,
            freq,
            mem;
    char volatile
            status;
    int     emm;
    pcmbuff start;
    unsigned long   size;
    pcmbuff volatile
            now;
    unsigned long   done;
}   PCMPCB;

typedef struct CH_CTRL {
    short   buf_seg,
            buf_off,
            buf_len,
            buf_ptr,
            buf_vdl,
            buf_int_cond,
```

```
            buf_int_off,
            buf_int_seg;
    char    key,
            length,
            gate_step,
            playing,
            work[12];
}   CH_CTRL;

typedef struct COMMON_WORK {
    CH_CTRL ch_ctrl[6];
    char    tempo,
            status_save,
            sb_work_2[62],
            sb_work_3[509];
    short   para_seg;
    char    sb_mode;
}   COMMON_WORK;

typedef struct EXT_COMMON_WORK {
    CH_CTRL ch_ctrl[4];
    short   rym_seg;
    char    status_save_2,
            sb_work_4[381];
}   EXT_COMMON_WORK;

typedef struct WORKBLOCK {
    COMMON_WORK common_work;
    char        driver_work[512];
    EXT_COMMON_WORK ext_common_work;
}   WORKBLOCK;

typedef struct PATTERN_DATA {
    unsigned char   length,
```

```
                    step,
                    data[1];
}   PATTERN_DATA;

typedef struct RYTMBLOCK {
    short           offset[128];
    PATTERN_DATA    pattern[1];
}   RYTMBLOCK;

/*
 * functions
 */
RESULT  check_pcmdrv ( void );
RESULT  start_pcmdrv ( void );
void    end_pcmdrv ( void );
int     call_pcmdrv ( int, union _REGS * );
RESULT  start_snddrv ( void );
void    end_snddrv ( void );
int     call_snddrv ( int, union _REGS *, int );
```

■ドライバを呼び出す(PCMDRV.C)

```
#include <stdio.h>
#include <dos.h>

#include "pcm.h"

#define BYTEPLAYBUFF    0

static WORKBLOCK __far  *Work;

RESULT  check_pcmdrv ( void ) {
    int fh;

    /* check driver is available or not ? */
    if ( ( fh = open ( DEVSNDDRV, 0 ) ) == EOF ) {
        return FAIL;
    }
    close ( fh );
    return SUCCESS;
}

RESULT  start_pcmdrv ( void ) {
    union   _REGS   regs;
    struct  _SREGS  sregs;
    int     i, seg, offset;
    char    *ptr;
    RYTMBLOCK __far *rym;

    /* allocate memory for FM/SSG driver */
    if ( _dos_allocmem ( sizeof ( WORKBLOCK ), &seg ) ) {
        return FAIL;
    }
    _FP_SEG ( Work ) = seg;
```

```
    _FP_OFF ( Work ) = 0;

    _segread ( &sregs );
    for ( i = 0; i < 6; ++i ) {
#if BYTEPLAYBUFF != 0
        ptr = (char *)malloc ( BYTEPLAYBUFF );
        /* don't care ptr is NULL or not */
        Work->common_work.ch_ctrl[i].buf_off = (int)ptr;
        Work->common_work.ch_ctrl[i].buf_seg = sregs.ds;
#endif
        Work->common_work.ch_ctrl[i].buf_len = BYTEPLAYBUFF;
    }
    for ( i = 0; i < 4; ++i ) {
#if BYTEPLAYBUFF != 0
        ptr = (char *)malloc ( BYTEPLAYBUFF );
        /* don't care ptr is NULL or not */
        Work->ext_common_work.ch_ctrl[i].buf_off = (int)ptr;
        Work->ext_common_work.ch_ctrl[i].buf_seg = sregs.ds;
#endif
        Work->ext_common_work.ch_ctrl[i].buf_len = BYTEPLAYBUFF;
    }
    if ( _dos_allocmem ( sizeof ( RYTMBLOCK ), &seg ) ) {
        return FAIL;
    }
    Work->ext_common_work.rym_seg = seg;
    FP_SEG ( rym ) = seg;
    FP_OFF ( rym ) = 0;
    offset = (char __far *)rym->pattern - (char __far *)rym;
    for ( i = 0; i < 128; ++i ) {
        rym->offset[i] = offset;
    }
    rym->pattern[0].length = 0;

    if ( call_snddrv ( CMDINITIALIZE2, &regs, FP_SEG(Work) ) ){
```

```
        return FAIL;
    }

    call_pcmdrv ( CMDINITDRV, &regs );
    atexit ( end_pcmdrv );
    atexit ( end_snddrv );
    return SUCCESS;
}

void    end_snddrv ( void ) {
    union   _REGS   regs;
    char    *ptr;
    int     i;

    /* terminate driver */
    regs.h.bl = 1;
    call_snddrv ( CMDCLEAR, &regs, 0 );

    /* free memory */
    _dos_freemem ( Work->ext_common_work.rym_seg );
    for ( i = 0; i < 6; ++i ) {
        ptr = (char *)Work->common_work.ch_ctrl[i].buf_off;
        if ( ptr != NULL ) free ( ptr );
    }
    for ( i = 0; i < 4; ++i ) {
        ptr = (char *)Work->ext_common_work.ch_ctrl[i].buf_off;
        if ( ptr != NULL ) free ( ptr );
    }
    _dos_freemem ( FP_SEG ( Work ) );
}

void    end_pcmdrv ( void ) {
    union   _REGS   regs;
```

```
    /* terminate driver */
    regs.h.bl = 1;
    call_pcmdrv ( CMDSTPTRN, &regs );
}

int call_snddrv ( int cmd, union _REGS *regs, int es ) {
    struct _SREGS   sregs;

    _segread ( &sregs );
    sregs.es = es;
    regs->h.ah = cmd;
    _int86x ( INTSNDDRV, regs, regs, &sregs );
    return regs->x.cflag;
}

int call_pcmdrv ( int cmd, union _REGS *regs ) {
    struct _SREGS   sregs;

    _segread ( &sregs );
    sregs.es = sregs.ds;
    regs->h.ah = cmd;
    _int86x ( INTPCMDRV, regs, regs, &sregs );
    return regs->x.cflag;
}
```

■ドライバを初期化する(INIT.C)

```
#include <stdio.h>
#include <dos.h>

#include "pcm.h"

/*
 * functions
 */
void    main ( void ) {
    if ( check_pcmdrv ( ) == FAIL ) {
        fprintf ( stderr, "No PCM driver installed¥n" );
        exit ( 1 );
    }
    if ( start_pcmdrv ( ) == FAIL ) {
        fprintf ( stderr, "Cannot start PCM driver¥n" );
        exit ( 1 );
    }
    exit ( 0 );
}
```

■ドライバの設定を表示する(SHOW.C)

```
#include <stdio.h>
#include <dos.h>

#include "pcm.h"

/*
 * data
 */
float   Freq[] = {
    44.10, 33.08, 22.05, 16.54, 11.03, 8.27, 5.52, 4.13
};

/*
 * functions
 */
void    main ( void ) {
    union _REGS   regs;
    int      i;
    char     *ptr;

    /* check driver */
    if ( check_pcmdrv ( ) == FAIL ) {
        fprintf ( stderr, "No PCM driver installed¥n" );
        exit ( 1 );
    }

    /* show volume */
    for ( i = 0; i < 5; ++i ) {
        regs.h.al = ( i << 4 );
        call_pcmdrv ( CMDGETVOL, &regs );
        printf ( "Vol%d: %3.0f%%    ", i + 1,
            (float)(regs.h.al & 0x0f ) / (float)15 * 100 );
```

```
    }
    putchar ( '¥n' );

    /* show mix. rate */
    call_pcmdrv ( CMDGETMIX, &regs );
    i = regs.h.al * 10;
    printf ( "Mixing rate: IN1:IN2=%02d%%:%02d%%¥n",
             100 - i, i );

    /* muting status */
    call_pcmdrv ( CMDGETMUTE, &regs );
    printf ( "Mute %s¥n", regs.h.al ? "ON" : "OFF" );

    /* sampling frequency */
    call_pcmdrv ( CMDGETFREQ, &regs );
    printf ( "Sampling freq.: %.2fKHz¥n", Freq[regs.h.al] );

    /* PCM mode */
    call_pcmdrv ( CMDGETPCM, &regs );

    printf ( "PCM mode: %d bits ",
             ( regs.h.al & 128 ) ? 16 : 8 );
    switch ( regs.h.al & 0x70 ) {
    case 0x00: ptr = "L-ch mono"; break;
    case 0x10: ptr = "R-ch mono"; break;
    case 0x20: ptr = "stereo"; break;
    }
    printf ( "%s¥n", ptr );

    printf ( "DSP input source: " );
    switch ( regs.h.al & 0x0f ) {
    case 0: ptr = "IN1:CD-DA    IN2:FM+LINE-IN"; break;
    case 1: ptr = "IN1:CD-DA    IN2:PCM-OUT"; break;
    case 2: ptr = "IN1:PCM-OUT  IN2:FM+LINE-IN"; break;
```

```
    case 3:
    case 4:
    case 5: ptr = "IN1:CD-DA"; break;
    case 6:
    case 7:
    case 8: ptr = "IN2:FM+LINE-IN"; break;
    }
    printf ( "%s¥n", ptr );

    printf ( "PCM input source: " );
    switch ( regs.h.al & 0x0f ) {
    case 0:
    case 1:
    case 2:
    case 3: ptr = "NONE"; break;
    case 4: ptr = "OUT1:DSP"; break;
    case 5: ptr = "OUT2:DSP"; break;
    case 6: ptr = "IN2:FM+LINE-IN"; break;
    case 7: ptr = "OUT1:DSP"; break;
    case 8: ptr = "OUT2:DSP"; break;
    }
    printf ( "%s¥n", ptr );
}
```

■入力ボリュームを調整する(VLM.C)

```
#include <stdio.h>
#include <dos.h>
#include <fcntl.h>
#include <io.h>
#include <ctype.h>

#include "pcm.h"

#define NUMVOL      5
#define MAXVOL      15

void    main ( int argc, char **argv ) {
    int   ch, vol;
    union _REGS regs;

    /* check driver */
    if ( check_pcmdrv ( ) == FAIL ) {
        fprintf ( stderr, "No PCM driver installed¥n" );
        exit ( 1 );
    }

    /* check arguments */
    if ( argc != 3 ) {
        fprintf ( stderr,
                  "usage:¥nvlm <volume-ch> <volume%>¥n" );
        exit ( 1 );
    }

    ch = atoi ( argv[1] );
    if ( ch <= 0 || ch > NUMVOL ) {
        fprintf ( stderr, "bad volume selector: %s¥n",
                  argv[1] );
```

```
        exit ( 1 );
    }
    --ch;                   /* 1 ... NUMVOL -> 0 ... NUMVOL-1 */

    vol = atoi ( argv[2] );
    if ( ! isdigit ( *argv[2] ) || vol < 0 || vol > 100 ) {
        fprintf ( stderr, "bad volume value: %s¥n", argv[2] );
        exit ( 1 );
    }
    vol = MAXVOL * vol / 100;

    /* set volume */
    regs.h.al = ( ch << 4 ) | vol;
    if ( call_pcmdrv ( CMDSETVOL, &regs ) ) {
        fprintf ( stderr, "Cannot set volume (%d)¥n",
                  regs.h.ah );
        exit ( 1 );
    }
    exit ( 0 );
}
```

■ミキシングレートを変更する(MIX.C)

```
#include <stdio.h>
#include <dos.h>
#include <fcntl.h>
#include <io.h>
#include <ctype.h>

#include "pcm.h"

void    main ( int argc, char **argv ) {
    int   rate;
    union _REGS regs;

    /* check driver */
    if ( check_pcmdrv ( ) == FAIL ) {
        fprintf ( stderr, "No PCM driver installed¥n" );
        exit ( 1 );
    }

    /* check arguments */
    if ( argc != 2 ) {
        fprintf ( stderr, "usage:¥nmix <IN1-rate>¥n" );
        exit ( 1 );
    }

    rate = atoi ( argv[1] );
    if ( ! isdigit ( *argv[1] ) || rate < 0 || rate > 100 ) {
        fprintf ( stderr, "bad mixing rate: %s¥n", argv[1] );
        exit ( 1 );
    }
    rate = 100 - rate;
    rate /= 10;         /* 0 ... 100 -> 0 ... 10 */
```

```
    /* set mixing rate */
    regs.h.al = rate;
    if ( call_pcmdrv ( CMDSETMIX, &regs ) ) {
        fprintf ( stderr, "Cannot set mix rate (%d)¥n",
                  regs.h.ah );
        exit ( 1 );
    }
    exit ( 0 );
}
```

■ミュート状態を変更する(MUTE.C)

```
#include <stdio.h>
#include <dos.h>
#include <fcntl.h>
#include <io.h>
#include <ctype.h>

#include "pcm.h"

void    main ( int argc, char **argv ) {
    int   sw;
    union _REGS regs;

    /* check driver */
    if ( check_pcmdrv ( ) == FAIL ) {
        fprintf ( stderr, "No PCM driver installed¥n" );
        exit ( 1 );
    }

    /* check arguments */
    if ( argc != 2 ) {
        fprintf ( stderr, "usage:¥nmute <on|off>¥n" );
        exit ( 1 );
    }
    if ( strcmp ( "on", argv[1] ) == 0 ) {
        sw = 1;
    } else if ( strcmp ( "off", argv[1] ) == 0 ) {
        sw = 0;
    } else {
        fprintf ( stderr, "bad selector: %s¥n", argv[1] );
        exit ( 1 );
    }
```

```
    /* set muting */
    regs.h.al = sw;
    if ( call_pcmdrv ( CMDSETMUTE, &regs ) ) {
        fprintf ( stderr, "Cannot set muting state (%d)¥n",
                  regs.h.ah );
        exit ( 1 );
    }
    exit ( 0 );
}
```

■サンプリングビット数を切り替える(BIT.C)

```
#include <stdio.h>
#include <dos.h>
#include <fcntl.h>
#include <io.h>
#include <ctype.h>

#include "pcm.h"

void    main ( int argc, char **argv ) {
    union _REGS  regs;

    /* check driver */
    if ( check_pcmdrv ( ) == FAIL ) {
        fprintf ( stderr, "No PCM driver installed¥n" );
        exit ( 1 );
    }
    /* get current mode */
    if ( call_pcmdrv ( CMDGETPCM, &regs ) ) {
        fprintf ( stderr, "Cannot get current mode (%d)¥n",
                  regs.h.ah );
        exit ( 1 );
    }

    /* check arguments */
    if ( argc != 2 ) {
        fprintf ( stderr, "usage:¥nbit <8|16>¥n" );
        exit ( 1 );
    }
    if ( strcmp ( "8", argv[1] ) == 0 ) {
        regs.h.al &= 0x7f;
    } else if ( strcmp ( "16", argv[1] ) == 0 ) {
        regs.h.al |= 0x80;
```

```
    } else {
        fprintf ( stderr, "bad bit: %s¥n", argv[1] );
        exit ( 1 );
    }

    /* set pcm mode */
    if ( call_pcmdrv ( CMDSETPCM, &regs ) ) {
        fprintf ( stderr, "Cannot set bit (%d)¥n", regs.h.ah );
        exit ( 1 );
    }
    exit ( 0 );
}
```

■サンプリングチャネル数を切り替える(CHAN.C)

```
#include <stdio.h>
#include <dos.h>
#include <fcntl.h>
#include <io.h>
#include <ctype.h>

#include "pcm.h"

void    main ( int argc, char **argv ) {
    union _REGS  regs;

    /* check driver */
    if ( check_pcmdrv ( ) == FAIL ) {
        fprintf ( stderr, "No PCM driver installed¥n" );
        exit ( 1 );
    }
    /* get current mode */
    if ( call_pcmdrv ( CMDGETPCM, &regs ) ) {
        fprintf ( stderr, "Cannot get current mode (%d)¥n",
                  regs.h.ah );
        exit ( 1 );
    }

    /* check arguments */
    if ( argc != 2 ) {
        fprintf ( stderr, "usage:¥nchan <l|r|s>¥n" );
        exit ( 1 );
    }
    regs.h.al &= 0x8f;
    if ( strcmp ( "l", argv[1] ) == 0 ) {
        regs.h.al |= 0;
    } else if ( strcmp ( "r", argv[1] ) == 0 ) {
```

```
        regs.h.al |= 0x10;
    } else if ( strcmp ( "s", argv[1] ) == 0 ) {
        regs.h.al |= 0x20;
    } else {
        fprintf ( stderr, "bad channel: %s¥n", argv[1] );
        exit ( 1 );
    }

    /* set pcm mode */
    if ( call_pcmdrv ( CMDSETPCM, &regs ) ) {
        fprintf ( stderr, "Cannot set bit (%d)¥n", regs.h.ah );
        exit ( 1 );
    }
    exit ( 0 );
}
```

■サンプリング周波数を切り替える(FREQ.C)

```
#include <stdio.h>
#include <dos.h>
#include <fcntl.h>
#include <io.h>
#include <ctype.h>

#include "pcm.h"

void    main ( int argc, char **argv ) {
    int   freq;
    union _REGS  regs;

    /* check driver */
    if ( check_pcmdrv ( ) == FAIL ) {
        fprintf ( stderr, "No PCM driver installed¥n" );
        exit ( 1 );
    }

    /* check arguments */
    if ( argc != 2 ) {
        fprintf ( stderr, "usage:¥nfreq <freq>¥n" );
        fprintf ( stderr,  "freq: 44.10, 33.08, 22.05, 16.54,
                  11.03, 8.27, 5.52, 4.13¥n" );
        exit ( 1 );
    }
    if ( strcmp ( "44.10", argv[1] ) == 0 ) {
        freq = 0;
    } else if ( strcmp ( "33.08", argv[1] ) == 0 ) {
        freq = 1;
    } else if ( strcmp ( "22.05", argv[1] ) == 0 ) {
        freq = 2;
    } else if ( strcmp ( "16.54", argv[1] ) == 0 ) {
```

```
        freq = 3;
    } else if ( strcmp ( "11.03", argv[1] ) == 0 ) {
        freq = 4;
    } else if ( strcmp ( "8.27", argv[1] ) == 0 ) {
        freq = 5;
    } else if ( strcmp ( "5.52", argv[1] ) == 0 ) {
        freq = 6;
    } else if ( strcmp ( "4.13", argv[1] ) == 0 ) {
        freq = 7;
    } else {
        fprintf ( stderr, "bad freq.: %s¥n", argv[1] );
        exit ( 1 );
    }

    /* set sampling freq. */
    regs.h.al = freq;
    if ( call_pcmdrv ( CMDSETFREQ, &regs ) ) {
        fprintf ( stderr, "Cannot set freq. rate (%d)¥n",
                  regs.h.ah );
        exit ( 1 );
    }
    exit ( 0 );
}
```

■入出力モードを切り替える(MODE.C)

```
#include <stdio.h>
#include <dos.h>
#include <fcntl.h>
#include <io.h>
#include <ctype.h>

#include "pcm.h"

void    main ( int argc, char **argv ) {
    int   mode;
    union _REGS  regs;

    /* check driver */
    if ( check_pcmdrv ( ) == FAIL ) {
        fprintf ( stderr, "No PCM driver installed¥n" );
        exit ( 1 );
    }
    /* get current mode */
    if ( call_pcmdrv ( CMDGETPCM, &regs ) ) {
        fprintf ( stderr, "Cannot get current mode (%d)¥n",
                  regs.h.ah );
        exit ( 1 );
    }

    /* check arguments */
    if ( argc != 2 ) {
        fprintf ( stderr, "usage:¥nmode <mode>¥n" );
        exit ( 1 );
    }
    mode = atoi ( argv[1] );
    if ( ! isdigit ( *argv[1] ) || mode < 0 || mode > 8 ) {
        fprintf ( stderr, "bad mode: %s¥n", argv[1] );
```

```
        exit ( 1 );
    }
    regs.h.al &= 0xf0;
    regs.h.al |= mode;

    /* set pcm mode */
    if ( call_pcmdrv ( CMDSETPCM, &regs ) ) {
        fprintf ( stderr, "Cannot set bit (%d)¥n", regs.h.ah );
        exit ( 1 );
    }
    exit ( 0 );
}
```

■サンプリングしファイルへ書き込む(REC.C)

```
#include <stdio.h>
#include <sys/types.h>
#include <dos.h>
#include <fcntl.h>
#include <io.h>

#include "pcm.h"

#define BUFFERSIZE  ((long)1024*256)
#define BLOCKSIZE   ((unsigned)1024*32)

void    main ( void ) {
    union _REGS  regs;
    PCMPCB  pcm;
    long    size;
    int     fh;
    unsigned    seg, len, result;
    char __far  *buffer;

    /* check driver */
    if ( check_pcmdrv ( ) == FAIL ) {
        fprintf ( stderr, "No PCM driver installed¥n" );
        exit ( 1 );
    }

    /* get current mode */
    if ( call_pcmdrv ( CMDGETPCM, &regs ) ) {
        fprintf ( stderr, "Cannot get current mode (%d)¥n",
                    regs.h.ah );
        exit ( 1 );
    }
    regs.h.al &= 0x0f;
```

```
    if ( regs.h.al < 3 ) {
        fprintf ( stderr, "bad mode: cannot sampling¥n" );
        exit ( 1 );
    }

    /* allocate memory for buffer */
    if ( _dos_allocmem ( (int)(BUFFERSIZE/16), &seg ) ) {
        fprintf ( stderr, "cannot allocate buffer¥n" );
        exit ( 1 );
    }
    FP_SEG ( buffer ) = seg;
    FP_OFF ( buffer ) = 0;

    /* make control block */
    pcm.mode = -1;              /* default mode */
    pcm.freq = -1;              /* default sampling rate */
    pcm.mem = MEMMAIN;          /* buffer is on main memory */
    pcm.start.addr = buffer;        /* buffer address */
    pcm.size = BUFFERSIZE;          /* buffer size */

    fprintf ( stderr, "Hit any key to start sampling¥n" );
    getchar ( );

    /* start PCM */
    regs.x.bx = (int)&pcm;
    if ( call_pcmdrv ( CMDPCMTRN, &regs ) ) {
        fprintf ( stderr, "Cannot start sampling (%x)¥n",
                  regs.h.ah );
        exit ( 1 );
    }

    /* wait to PCM done */
    while ( pcm.status == PCMRUNNING ) {
        fprintf ( stderr, "¥r%8ld%% buffer left",
```

```
            ( BUFFERSIZE - pcm.done ) * 100 / BUFFERSIZE );
        fflush ( stderr );
        if ( kbhit ( ) ) {
            call_pcmdrv ( CMDSTPTRN, &regs );
            fprintf ( stderr, "¥raborted                ¥n" );
            exit ( 1 );
        }
        sleep ( 2 );
    }
    fprintf ( stderr, "¥rdone(%d)                ¥n",
              pcm.status );

    /* flush data to file */
    if ( _dos_creat ( "pcm.dat", _O_WRONLY, &fh ) ) {
        fprintf ( stderr, "cannot create data file¥n" );
        exit ( 1 );
    }
    for ( size = BUFFERSIZE; size > 0;
          size -= (long)BLOCKSIZE ) {
        len = (size < (long)BLOCKSIZE) ? (int)size : BLOCKSIZE;
        _dos_write ( fh, buffer, len, &result );
        buffer += len;
    }
    _dos_close ( fh );
}
sleep ( int sec ) {
    time_t  now, when;

    time ( &when );
    when += sec;
    do {
        time ( &now );
    } while ( now < when );
}
```

■データファイルを再生する(PLAY.C)

```
#include <stdio.h>
#include <sys/types.h>
#include <dos.h>
#include <fcntl.h>
#include <io.h>

#include "pcm.h"

#define BUFFERSIZE  ((long)1024*256)
#define BLOCKSIZE   ((unsigned)1024*32)

void    main ( void ) {
    union _REGS  regs;
    PCMPCB  pcm;
    long    size;
    int     fh;
    unsigned    seg, len, result;
    char __far  *buffer, __far *ptr;

    /* check driver */
    if ( check_pcmdrv ( ) == FAIL ) {
        fprintf ( stderr, "No PCM driver installed¥n" );
        exit ( 1 );
    }

    /* get current mode */
    if ( call_pcmdrv ( CMDGETPCM, &regs ) ) {
        fprintf ( stderr, "Cannot get current mode (%d)¥n",
                    regs.h.ah );
        exit ( 1 );
    }
```

```
    regs.h.al &= 0x0f;
    if ( regs.h.al > 2 ) {
        fprintf ( stderr, "bad mode: cannot playing¥n" );
        exit ( 1 );
    }

    /* allocate memory for buffer */
    if ( _dos_allocmem ( (int)(BUFFERSIZE/16), &seg ) ) {
        fprintf ( stderr, "cannot allocate buffer¥n" );
        exit ( 1 );
    }
    FP_SEG ( buffer ) = seg;
    FP_OFF ( buffer ) = 0;

    /* read from file */
    if ( _dos_open ( "pcm.dat", _O_RDONLY, &fh ) ) {
        fprintf ( stderr, "cannot open data file¥n" );
        exit ( 1 );
    }
    ptr = buffer;
    for ( size = BUFFERSIZE; size > 0; size -= (long)result ) {
        len = (size < (long)BLOCKSIZE) ? (int)size : BLOCKSIZE;
        _dos_read ( fh, ptr, len, &result );
        if ( result == 0 ) break;
        ptr += result;
    }
    _dos_close ( fh );

    /* make control block */
    pcm.mode = -1;                  /* default mode */
    pcm.freq = -1;                  /* default sampling rate */
    pcm.mem = MEMMAIN;              /* buffer is on main memory */
    pcm.start.addr = buffer;          /* buffer address */
    pcm.size = BUFFERSIZE - size;     /* buffer size */
```

```
    /* start PCM */
    regs.x.bx = (int)&pcm;
    if ( call_pcmdrv ( CMDPCMTRN, &regs ) ) {
        fprintf ( stderr, "Cannot start playing (%x)¥n",
                  regs.h.ah );
        exit ( 1 );
    }

    /* wait to PCM done */
    while ( pcm.status ==  PCMRUNNING ) {
        fprintf ( stderr, "¥r%8ld%% buffer left",
            ( pcm.size - pcm.done ) * 100 / pcm.size );
        fflush ( stderr );
        if ( kbhit ( ) ) {
           call_pcmdrv ( CMDSTPTRN, &regs );
           fprintf ( stderr, "¥raborted                  ¥n" );
           exit ( 1 );
        }
        sleep ( 2 );
    }
    fprintf ( stderr, "¥rdone(%d)               ¥n",
            pcm.status );
}

sleep ( int sec ) {
    time_t        now, when;

    time ( &when );
    when += sec;
    do {
        time ( &now );
    } while ( now < when );
}
```